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”

Виконав студент групи КІ-307:

Луцик Н.І.

Перевірив:

старший викладач каф. ЕОМ

Козак Н.Б.

Львів-2024

**ЗАВДАННЯ НА КУРСОВИЙ ПРОЄКТ**

1. Цільова мова транслятора – мова програмування С або асемблер для 32/64 розрядного процесора.

2. Для отримання виконавчого файлу на виході розробленого транслятора скористатися середовищем Microsoft Visual Studio бо будь-яким іншим.

3. Мова розробки транслятора: С/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 проєктування:**

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:**

* Тип даних: Int32
* Блок тіла програми: StartProgram; StartBlok Variable…; EndBlok
* Оператор вводу: Scan ()
* Оператор виводу: Print ()
* Оператори: If-else (C)

Goto (C)

For-To (Паскаль)

For-Downto (Паскаль)

While (Бейсік)

Repeat-Until (Паскаль)

* Регістр ключових слів: Up-Low перший символ Up
* Регістр ідентифікаторів: Low-Up2 перший символ \_
* Операції арифметичні: ++, --, \*\*, Div, Mod
* Операції порівняння: Eq, Neq, Ls, Gr
* Операції логічні: Not, And, Or
* Коментар: !! …!!
* Ідентифікатори змінних, числові константи
* Оператор присвоєння: <-

АНОТАЦІЯ

У цьому курсовому проєкті створено транслятор, який здійснює перетворення вхідної мови, визначеної варіантом завдання, у мову асемблера. Трансляція включає три ключові етапи: лексичний аналіз, синтаксичний аналіз та генерацію коду.

Під час лексичного аналізу вхідний потік символів розбивається на лексеми, що заносяться до спеціальної таблиці. Для кожної лексеми генерується унікальний числовий ідентифікатор, що спрощує подальшу обробку. До таблиці також додається додаткова інформація: номер рядка, значення (у разі числової лексеми) та інші важливі параметри.

Синтаксичний аналіз здійснюється за допомогою висхідного методу без повернення, що дозволяє побудувати дерево розбору, просуваючись від листків до кореня. Цей етап забезпечує правильне структурування даних відповідно до синтаксичних правил мови.

На етапі генерації коду обробляється таблиця лексем для створення асемблерного коду, що відповідає кожному блоку. Згенерований код записується у вихідний файл і готовий до компіляції.

Отриманий асемблерний код можна скомпілювати та виконати за допомогою інструментів, таких як LINK, ML тощо.

**ЗМІСТ**

[АНОТАЦІЯ 5](#_Toc188415215)

[2. Формальний опис вхідної мови програмування 9](#_Toc188415216)

[2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура. 9](#_Toc188415217)

[3. Розробка транслятора з вхідної мови програмування 14](#_Toc188415218)

[3.1. Вибір технології програмування. 14](#_Toc188415219)

[3.2. Проектування таблиць транслятора та вибір структур даних. 14](#_Toc188415220)

[3.3. Розробка лексичного аналізатора. 16](#_Toc188415221)

[**3.3.1.** **Розробка алгоритму роботи лексичного аналізатора.** 17](#_Toc188415222)

[**3.3.2.** **Опис програми реалізації лексичного аналізатора.** 20](#_Toc188415223)

[3.4. Розробка синтаксичного та семантичного аналізатора. 21](#_Toc188415224)

[**3.4.1.** **Розробка дерева граматичного розбору.** 23](#_Toc188415225)

[**3.4.2.** **Розробка алгоритму роботи синтаксичного і семантичного аналізатора.** 24](#_Toc188415226)

[**3.4.3.** **Опис програми реалізації синтаксичного та семантичного аналізатора.** 26](#_Toc188415227)

[3.5. Розробка генератора коду. 30](#_Toc188415228)

[**3.5.1.** **Розробка алгоритму роботи генератора коду.** 30](#_Toc188415229)

[**3.5.2.** **Опис програми реалізації генератора коду.** 32](#_Toc188415230)

[4. Налагодження та тестування розробленого транслятора 37](#_Toc188415231)

[4.1. Опис інтерфейсу та інструкції користувачу. 38](#_Toc188415232)

[4.2. Виявлення лексичних і синтаксичних помилок. 38](#_Toc188415233)

[4.3. Перевірка роботи транслятора за допомогою тестових задач. 40](#_Toc188415234)

[Висновки 43](#_Toc188415235)

[Список літературних джерел 44](#_Toc188415236)

[Додатки 45](#_Toc188415237)

**ВСТУП**

Термін «транслятор» відноситься до програмного забезпечення, яке здійснює перетворення вихідного коду, написаного однією мовою програмування, у еквівалентний код іншої. Якщо вхідна мова є мовою високого рівня, а вихідна — мова асемблера або машинний код, такий транслятор називається компілятором.

Транслятори поділяються на два основних види: компілятори та інтерпретатори. Процес компіляції включає два основні етапи: аналіз і синтез. На етапі аналізу вихідний код розбивається на лексеми, перевіряється його синтаксична правильність і створюється проміжне представлення програми. Під час синтезу це представлення перетворюється на об'єктний код, який складається з машинних інструкцій і може виконуватися безпосередньо на комп’ютері.

Інтерпретатори, на відміну від компіляторів, не створюють окремого виконуваного файлу. Вони аналізують програму, формують її проміжне представлення, але не генерують об'єктного коду. Інтерпретатор виконує команди програми безпосередньо під час її обробки.

Компілятор дозволяє перетворювати вихідний код з однієї мови програмування на іншу. Вхідними даними для компілятора є текст вихідної програми, а результатом — об'єктний код, адаптований для конкретної апаратної платформи. Варто зазначити, що сам компілятор може бути створений на третій мові програмування, що демонструє його універсальність і гнучкість у розробці програмного забезпечення.

1. **ОГЛЯД МЕТОДІВ ТА СПОСОБІВ ПРОЄКТУВАННЯ ТРАНСЛЯТОРІВ**

Транслятор — це програмне забезпечення, яке перетворює вихідний код, написаний на одній мові програмування, у робочий код, представлений об’єктною мовою. До трансляторів належать різні види програм, кожна з яких має свої особливості у процесі трансляції. Сучасні транслятори поділяються на три основні категорії: асемблери, компілятори та інтерпретатори.

**Асемблер** — це інструмент, який перетворює символічний код на машинні інструкції. Його ключова особливість полягає в тому, що кожна команда символічного коду безпосередньо відповідає одній машинній інструкції.

**Компілятор** — це програма, яка транслює вихідний код з мови високого рівня у машинний код. У порівнянні з асемблером, компілятор виконує складніше перетворення, адаптуючи код для конкретної архітектури.

**Інтерпретатор** — це програма, що послідовно виконує команди вихідного коду. На відміну від компілятора, інтерпретатор не створює окремого виконуваного файлу, а обробляє інструкції під час виконання, що полегшує процес тестування і налагодження.

Процес трансляції включає кілька етапів:

* **Лексичний аналіз**. Вхідний код розбивається на лексеми, що представляють окремі слова та символи мови програмування. На цьому етапі також виявляються помилки, наприклад, некоректні символи чи неправильний формат ідентифікаторів.
* **Синтаксичний аналіз**. Створюється синтаксичне дерево, яке відображає структуру програми відповідно до граматики мови. Для цього використовуються методи аналізу, такі як LL(1) чи LR(1).
* **Семантичний аналіз**. Перевіряються логічні залежності між елементами програми, які не описуються синтаксисом. Це включає перевірку типів даних, областей видимості та правильність параметрів функцій.
* **Оптимізація коду**. Поліпшується ефективність виконання програми. Оптимізація може бути локальною чи глобальною, а також залежною чи незалежною від конкретної апаратної архітектури.
* **Генерація коду**. Завершується створенням об’єктного або асемблерного коду, готового для подальшої компіляції чи виконання.

Фази трансляції можуть поєднуватися або бути скороченими залежно від особливостей реалізації транслятора. Наприклад, у простих однопрохідних трансляторах проміжне представлення та оптимізація можуть бути відсутні, а інші етапи часто об'єднуються.

На етапі лексичного аналізу формується таблиця, яка містить ідентифікатори, рядки та числові значення. Синтаксичний аналіз генерує дерево розбору, що використовується для оптимізації та генерації коду. Контекстний аналіз забезпечує коректність типів, контроль областей видимості та відповідність параметрів функцій.

Результатом всіх етапів трансляції є об’єктний код, оптимізований для конкретної апаратної архітектури, який може бути скомпільований і виконаний на цільовій платформі. Кожен етап процесу забезпечує послідовне перетворення вихідного коду у робочий виконуваний файл, готовий до запуску.

1. Формальний опис вхідної мови програмування
   1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура.

labeled\_point = label , ":"

goto\_label = tokenGOTO, label, ";"

program\_name = ident,";"

value\_type = tokenINTEGER16

other\_declaration\_ident = tokenCOMMA , ident

declaration = value\_type , ident , {other\_declaration\_ident}

unary\_operator = tokenNOT | tokenMINUS | tokenPLUS

unary\_operation = unary\_operator , expression

binary\_operator = tokenAND | tokenOR | tokenEQUAL | tokenNOTEQUAL | tokenLESSOREQUAL | tokenGREATEROREQUAL | tokenPLUS | tokenMINUS | tokenMUL | tokenDIV | tokenMOD

binary\_action = binary\_operator , expression

left\_expression = group\_expression | unary\_operation | ident | value

expression = left\_expression , {binary\_action}

group\_expression = tokenGROUPEXPRESSIONBEGIN , expression , tokenGROUPEXPRESSIONEND

//

bind\_right\_to\_left = ident , tokenRLBIND , expression

bind\_left\_to\_right = expression , tokenLRBIND , ident

//

if\_expression = expression

body\_for\_true = {statement} , ";"

body\_for\_false = tokenELSE , {statement} , ";"

cond\_block = tokenIF , tokenGROUPEXPRESSIONBEGIN , if\_expression , tokenGROUPEXPRESSIONEND , body\_for\_true , [body\_for\_false];

//

cycle\_begin\_expression = expression

cycle\_counter = ident

cycle\_counter\_rl\_init = cycle\_counter , tokenRLBIND , cycle\_begin\_expression

cycle\_counter\_lr\_init = cycle\_begin\_expression , tokenLRBIND , cycle\_counter

cycle\_counter\_init = cycle\_counter\_rl\_init | cycle\_counter\_lr\_init

cycle\_counter\_last\_value = value

cycle\_body = tokenDO , statement , {statement}

forto\_cycle = tokenFOR , cycle\_counter\_init , tokenTO , cycle\_counter\_last\_value , cycle\_body , ";"

continue\_while = tokenCONTINUE , tokenWHILE

exit\_while = tokenEXIT , tokenWHILE

statement\_in\_while\_body = statement | continue\_while | exit\_while

while\_cycle\_head\_expression = expression

while\_cycle = tokenWHILE , while\_cycle\_head\_expression , {statement\_in\_while\_body} , tokenEND , tokenWHILE

//

repeat\_until\_cycle\_cond = group\_expression

repeat\_until\_cycle = tokenREPEAT , {statement} , tokenUNTIL , repeat\_until\_cycle\_cond

input = tokenGET , tokenGROUPEXPRESSIONBEGIN , ident , tokenGROUPEXPRESSIONEND

output = tokenPUT , tokenGROUPEXPRESSIONBEGIN , expression , tokenGROUPEXPRESSIONEND

statement = bind\_right\_to\_left | bind\_left\_to\_right | cond\_block | forto\_cycle | while\_cycle | repeat\_until\_cycle | labeled\_point | goto\_label | input | output

program = tokenNAME , program\_name , tokenSEMICOLON , tokenBODY , tokenDATA , [declaration] , tokenSEMICOLON , {statement} , tokenEND

//

digit = digit\_0 | digit\_1 | digit\_2 | digit\_3 | digit\_4 | digit\_5 | digit\_6 | digit\_7 | digit\_8 | digit\_9

non\_zero\_digit = digit\_1 | digit\_2 | digit\_3 | digit\_4 | digit\_5 | digit\_6 | digit\_7 | digit\_8 | digit\_9

unsigned\_value = ((non\_zero\_digit , {digit}) | digit\_0)

value = [sign] , unsigned\_value

// -- hello wolrd

letter\_in\_lower\_case = a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

letter\_in\_upper\_case = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

ident = tokenUNDERSCORE , letter\_in\_upper\_case , letter\_in\_upper\_case , letter\_in\_upper\_case , letter\_in\_upper\_case , letter\_in\_upper\_case , letter\_in\_upper\_case , letter\_in\_upper\_case

label = letter\_in\_lower\_case , {letter\_in\_lower\_case}

//

sign = sign\_plus | sign\_minus

sign\_plus = '-'

sign\_minus = '+'

//

digit\_0 = '0'

digit\_1 = '1'

digit\_2 = '2'

digit\_3 = '3'

digit\_4 = '4'

digit\_5 = '5'

digit\_6 = '6'

digit\_7 = '7'

digit\_8 = '8'

digit\_9 = '9'

//

tokenCOLON = ":"

tokenGOTO = "Goto"

tokenINTEGER16 = "Int32"

tokenCOMMA = ","

tokenNOT = "Not"

tokenAND = "And"

tokenOR = "Or"

tokenEQUAL = "Eq"

tokenNOTEQUAL = "Neq"

tokenPLUS = "++"

tokenMINUS = "--"

tokenMUL = "\*\*"

tokenDIV = "Div"

tokenMOD = "Mod"

tokenGROUPEXPRESSIONBEGIN = "("

tokenGROUPEXPRESSIONEND = ")"

tokenRLBIND = "<-"

tokenLRBIND = ","

tokenELSE = "else"

tokenIF = "If"

tokenDO = "Do"

tokenFOR = "For"

tokenTO = "To"

tokenWHILE = "While"

tokenCONTINUE = "Continue"

tokenEXIT = "Exit"

tokenREPEAT = "Repeat"

tokenUNTIL = "Until"

tokenGET = "Scan"

tokenPUT = "Print"

tokenNAME = "StartProgram"

tokenBODY = "StartBlok"

tokenDATA = "Variable"

tokenEND = "EndBlok"

tokenSEMICOLON = ";"

//

tokenUNDERSCORE = "\_"

//

A = "A"

B = "B"

C = "C"

D = "D"

E = "E"

F = "F"

G = "G"

H = "H"

I = "I"

J = "J"

K = "K"

L = "L"

M = "M"

N = "N"

O = "O"

P = "P"

Q = "Q"

R = "R"

S = "S"

T = "T"

U = "U"

V = "V"

W = "W"

X = "X"

Y = "Y"

Z = "Z"

//

a = "a"

b = "b"

c = "c"

d = "d"

e = "e"

f = "f"

g = "g"

h = "h"

i = "i"

j = "j"

k = "k"

l = "l"

m = "m"

n = "n"

o = "o"

p = "p"

q = "q"

r = "r"

s = "s"

t = "t"

u = "u"

v = "v"

w = "w"

x = "x"

y = "y"

z = "z"

1. Розробка транслятора з вхідної мови програмування
   1. Вибір технології програмування.

Перед тим як розпочинати створювати програму, для більш швидкого і ефективного її написання, необхідно розробити алгоритм її функціонування, та вибрати технологію програмування, середовище програмування.

Для виконання поставленого завдання найбільш доцільно буде використати середовище програмування Microsoft Visual Studio 2022, та мову програмування C/С++.

Для якісного і зручного використання розробленої програми користувачем, було прийнято рішення створення консольного інтерфейсу.

* 1. Проектування таблиць транслятора та вибір структур даних.

**Опис структури LexemInfo**

LexemInfo — це структура, яка використовується для зберігання інформації про лексеми, отримані під час лексичного аналізу. Вона надає публічний доступ до своїх членів і створена для зручного доступу до властивостей кожної лексеми. Детальний опис її елементів та функцій наведено нижче.

**Члени структури:**

1. **char lexemStr[MAX\_LEXEM\_SIZE]**  
   Масив символів, у якому зберігається лексема у вигляді рядка.  
   MAX\_LEXEM\_SIZE визначає максимальну довжину лексеми, зазвичай задається як константа.
2. **unsigned long long int lexemId**  
   Унікальний ідентифікатор лексеми, який дозволяє відрізняти її від інших.
3. **unsigned long long int tokenType**  
   Тип токена, до якого належить лексема (наприклад, константа, оператор чи ключове слово).
4. **unsigned long long int ifvalue**  
   Додаткове значення, яке використовується в умовних виразах або для контекстуального аналізу.  
   Наприклад, це може бути значення для порівнянь чи виконання умов.
5. **unsigned long long int row**  
   Номер рядка у вихідному коді, де знаходиться лексема. Це корисно для діагностики або виведення повідомлень про помилки.
6. **unsigned long long int col**  
   Номер колонки у рядку, де розташована лексема.
7. **Коментар або місце для розширення**  
   Позначка // TODO: ... передбачає можливість додавання нових полів для розширення функціональності структури.

**Конструктори:**

1. **Конструктор за замовчуванням: LexemInfo()**  
   Ініціалізує всі поля структури стандартними значеннями (наприклад, нулями чи порожніми рядками).
2. **Параметризований конструктор: LexemInfo(const char\* lexemStr, unsigned long long int lexemId, unsigned long long int tokenType, unsigned long long int ifvalue, unsigned long long int row, unsigned long long int col)**  
   Створює об’єкт з переданими значеннями:
   * lexemStr — рядок лексеми.
   * lexemId — унікальний ідентифікатор.
   * tokenType — тип токена.
   * ifvalue — додаткове значення.
   * row і col — позиція лексеми у вихідному коді.
3. **Конструктор копіювання: LexemInfo(const NonContainedLexemInfo& nonContainedLexemInfo)**  
   Ініціалізує об'єкт на основі іншої структури NonContainedLexemInfo, дозволяючи зберегти сумісність з іншими форматами даних.

**Призначення:**

Структура LexemInfo забезпечує:

* Збереження інформації про токени під час лексичного аналізу.
* Легкий доступ до позиції лексеми (рядок і колонка) для генерації звітів чи повідомлень про помилки.
* Структурування даних, необхідних для побудови синтаксичного дерева.
* Розширення для семантичного аналізу чи інших етапів компіляції.
  1. Розробка лексичного аналізатора.

Метою лексичного аналізу є розбиття вхідного тексту, який складається з послідовності символів, на окремі слова або лексеми. Це передбачає виокремлення слів із суцільної послідовності символів. Усі символи вхідного тексту поділяються на дві категорії: ті, що є частинами лексем, і ті, що розділяють лексеми.

Для цього зазвичай використовуються стандартні методи обробки рядків. Програма аналізується послідовно від початку до кінця. Основні елементи тексту, або лексичні одиниці, розділяються пробілами, знаками операцій та спеціальними символами (наприклад, переведення рядка чи табуляція). Таким чином визначаються та розпізнаються:

* **Ідентифікатори**
* **Літерали**
* **Термінальні символи** (зокрема операції та ключові слова).

Після виділення лексеми вона розпізнається та заноситься до таблиці лексем за допомогою унікального номера, який відповідає кожній лексемі із загального набору. Це дозволяє наступним фазам компіляції працювати з лексемами не як із послідовностями символів, а як із їх унікальними номерами. Такий підхід значно спрощує роботу синтаксичного аналізатора, зокрема:

* Полегшує перевірку належності лексеми до синтаксичної конструкції.
* Дозволяє зручно переглядати програму як вперед, так і назад від поточної позиції аналізу.

Окрім унікального номера, таблиця лексем також містить інформацію про рядок, у якому знаходиться відповідна лексема (це корисно для вказівки місця помилки), а також інші додаткові дані.

Під час лексичного аналізу коментарі відкидаються, оскільки вони не впливають на виконання програми, синтаксичний розбір чи генерацію коду.

**Типи лексем (лексичні класи):**

1. **Ключові слова**:  
   StartProgram, Variable, EndBlok, Scan, Print, Int32, If, else, For, Goto, Downto, Repeat, Until, While.
2. **Ідентифікатори**.
3. **Числові константи**:  
   Цілі числа без знаку.
4. **Оператор присвоєння**:  
   <-.
5. **Знаки операцій**:  
   ++, --, \*\*, Div, Gr, Ls, Eq, Neq, Not, And, Or.
6. **Роздільники**:  
   ; ,
7. **Дужки**:  
   (, )
   * 1. **Розробка алгоритму роботи лексичного аналізатора.**

Даний лексичний аналізатор — це програмний модуль, який перетворює вхідний текст на послідовність лексем (основних синтаксичних одиниць) і класифікує їх за типами. Основна мета аналізатора — підготувати текст для подальших етапів аналізу, таких як синтаксичний або семантичний. У коді реалізовано функціонал для розпізнавання ключових слів, значень, ідентифікаторів, а також для обробки коментарів.

Як працює аналізатор:

**1. Основні структури даних**

**LexemInfo**

Містить інформацію про кожну лексему:

* **lexemStr** — текстовий рядок лексеми.
* **lexemId** — унікальний ідентифікатор лексеми.
* **tokenType** — тип токена (ключове слово, ідентифікатор, значення тощо).
* **ifvalue** — додаткова інформація для значень.
* **row і col** — позиція лексеми в тексті (номер рядка та стовпця).
* **NonContainedLexemInfo** — служить для тимчасового зберігання лексем, забезпечуючи використання буфера (tempStrFor\_123).

**2. Основні масиви**

* **lexemesInfoTable** — таблиця, де зберігаються всі знайдені лексеми.
* **identifierIdsTable** — таблиця для збереження ідентифікаторів, яка запобігає дублюванню.

Цей лексичний аналізатор — програмний модуль, який розбиває вхідний текст на лексеми (синтаксичні одиниці) та класифікує їх за типами. Його основна задача — підготувати текст для подальшого синтаксичного та семантичного аналізу. Аналізатор має низку функцій для розпізнавання ключових слів, значень, ідентифікаторів, а також для обробки коментарів. Ось як він працює:

**Структури даних**

1. **LexemInfo** Зберігає інформацію про кожну лексему:
   * lexemStr — текст лексеми.
   * lexemId — унікальний ідентифікатор.
   * tokenType — тип токена (ключове слово, ідентифікатор, значення тощо).
   * ifvalue — додаткова інформація для числових значень.
   * row і col — позиція лексеми в тексті (рядок і стовпець).
2. **NonContainedLexemInfo** Використовується для тимчасового збереження лексем у буфері (наприклад, tempStrFor\_123).

**Основні масиви**

* lexemesInfoTable — містить усі знайдені лексеми.
* identifierIdsTable — зберігає ідентифікатори, запобігаючи дублюванню.

Етапи роботи

**1. Токенізація (tokenize)**

Функція розбиває текст на токени за допомогою регулярного виразу:

* Регулярний вираз (TOKENS\_RE) визначає правила формування токенів (ключові слова, числа, ідентифікатори тощо).
* Знайдені токени обробляються по черзі за допомогою ітератора (std::sregex\_token\_iterator).

**2. Ідентифікація токенів**

Для кожного токена викликаються функції:

* tryToGetKeyWord — перевіряє, чи є токен ключовим словом.
* tryToGetIdentifier — визначає, чи це ідентифікатор.
* tryToGetUnsignedValue — перевіряє, чи це числове значення.

Токени, які не відповідають жодному з цих шаблонів, позначаються як "непередбачувані лексеми".

**3. Обробка лексем**

* **Ключові слова** розпізнаються за допомогою регулярного виразу (KEYWORDS\_RE) та отримують унікальний lexemId.
* **Ідентифікатори** перевіряються за шаблоном (IDENTIFIERS\_RE) і додаються до identifierIdsTable.
* **Числові значення** розпізнаються регулярним виразом (UNSIGNEDVALUES\_RE) і зберігаються в полі ifvalue.

**4. Видалення коментарів**

Функція commentRemover видаляє коментарі:

* Однорядкові (//).
* Багаторядкові (/\* ... \*/). Коментарі замінюються пробілами, щоб зберегти структуру тексту.

**5. Визначення позицій лексем**

Функція setPositions встановлює рядок і стовпець для кожної лексеми, дозволяючи визначати точне місце розташування помилок у тексті.

**6. Формування результатів**

Результати аналізу подаються у вигляді таблиці(табл.1), яка включає:

* **Індекс лексеми** — номер у загальному списку.
* **Текст лексеми** — її оригінальний вигляд.
* **Ідентифікатор** — унікальний номер для кожного типу.
* **Тип лексеми** — ключове слово, ідентифікатор, значення тощо.
* **Значення** — числове значення для числових лексем.
* **Рядок і стовпець** — позиція у тексті.

**Табл.1.**

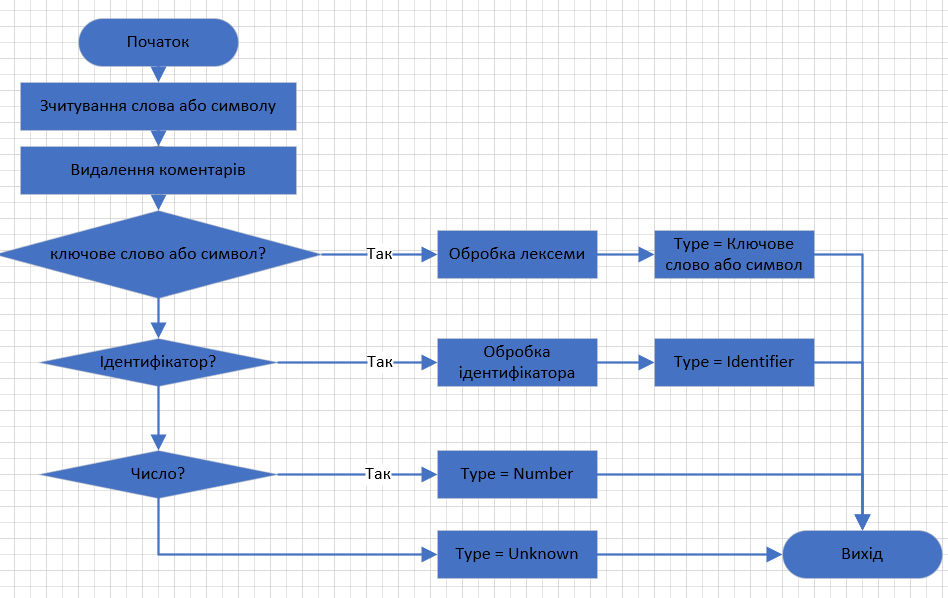
| **Індекс** | **Текст лексеми** | **Ідентифікатор** | **Тип** | **Значення** | **Рядок** | **Стовпець** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | StartProgram | 101 | Ключове слово | - | 1 | 1 |
| 1 | \_p | 1 | Ідентифікатор | - | 1 | 9 |
| 2 | StartBlok | 102 | Ключове слово | - | 2 | 1 |
| 3 | 123 | 1001 | Значення | 123 | 3 | 5 |
| 4 | EndBlok | 103 | Ключове слово | - | 4 | 1 |

**Особливості**

1. **Буферизація**: Використання буфера (tempStrFor\_123) для оптимізації пам’яті.
2. **Гнучкість**: Регулярні вирази (TOKENS\_RE, IDENTIFIERS\_RE тощо) можна легко адаптувати.
3. **Обробка помилок**: Некоректні токени автоматично позначаються як помилкові, не впливаючи на решту аналізу.

Переваги

* **Простота аналізу**: Легко виявляти помилки та несподівані лексеми завдяки координатам у тексті.
* **Універсальність**: Структурована таблиця полегшує додавання нових типів лексем.
* **Зручність**: Вся інформація про лексеми зберігається в єдиній систематизованій формі.



*Рис. 3.1. Граф-схема алгоритму роботи лексичного аналізатора.*

* + 1. **Опис програми реалізації лексичного аналізатора.**

Основна мета лексичного аналізатора — розділити вхідний текст програми на лексеми, тобто окремі смислові одиниці, які використовуються для подальшого аналізу. Символи у тексті поділяються на ті, що належать до лексем, і роздільники. Аналіз виконується послідовно, від початку до кінця тексту, з використанням стандартних методів обробки рядків. Базові елементи (лексичні одиниці) визначаються за допомогою пробілів, знаків операцій та спеціальних символів (наприклад, переведення рядка або табуляція). У результаті аналізатор розпізнає ідентифікатори, літерали та термінальні символи (наприклад, оператори чи ключові слова).

Робота програми триває доти, доки не буде оброблено весь текст. Для цього використовується функція tokenize(), яка зчитує вміст файлу, виділяє лексеми та порівнює їх із зарезервованими словами. У разі збігу лексема отримує відповідний тип або значення (наприклад, для числових констант).

Всі розпізнані лексеми додаються до списку m\_tokens з унікальним типом. Це дає можливість подальшим фазам компіляції працювати з лексемами як із конкретними типами, а не послідовностями символів, що значно полегшує синтаксичний аналіз. Такий підхід дозволяє легко перевіряти відповідність лексеми синтаксичним конструкціям або здійснювати навігацію текстом програми (вперед і назад від поточної позиції). У таблиці лексем також зберігаються координати лексем (номер рядка і стовпця), що спрощує пошук місця помилки. Крім того, додається метаінформація для подальших етапів аналізу.

Під час аналізу також виявляються лексичні помилки, наприклад, некоректні символи або неправильні ідентифікатори. Вони ігноруються, оскільки не впливають на синтаксичний розбір чи генерацію коду.

У рамках цього проєкту реалізовано прямий лексичний аналізатор, який виділяє лексеми з тексту програми та створює таблицю лексем для подальшої обробки.

* 1. Розробка синтаксичного та семантичного аналізатора.

Синтаксичний аналіз – це процес, що визначає, чи належить деяка послідовність лексем граматиці мови програмування. В принципі, для будь-якої граматики можна побудувати синтаксичний аналізатор, але граматики, які використовуються на практиці, мають спеціальну форму. Наприклад, відомо, що для будь-якої контекстно-вільної граматики може бути побудований аналізатор, складність якого не перевищує O(n3) для вхідного рядка довжиною n.

Код реалізує лексичний і синтаксичний аналізатор із побудовою абстрактного синтаксичного дерева (AST) на основі методу Кока-Янгера-Касамі (CYK) та рекурсивного спуску. Розглянемо основні етапи роботи:

**1. Лексичний аналіз**

Лексичний аналізатор розбиває вхідний текст на лексеми (мінімальні значущі одиниці мови, такі як ідентифікатори, ключові слова, константи тощо) та зберігає їх у таблиці LexemInfo.

**2. Метод CYK для синтаксичного аналізу**

* **Ініціалізація**: створюється таблиця parseInfoTable, де кожна комірка містить множину символів граматики.
* **Заповнення таблиці**: використовується двовимірний підхід, де кожна комірка заповнюється на основі правил граматики:
  + Якщо правило має один елемент справа, перевіряється відповідність лексеми цьому правилу.
  + Якщо правило має два елементи, шукається розбиття, яке дозволяє побудувати комбінацію двох піддерев.
* Після завершення побудови таблиці перевіряється наявність стартового символу граматики у верхньому правому куті таблиці. Якщо символ є, аналіз вважається успішним.

**3. Рекурсивний спуск**

Якщо метод CYK не успішний або обрано режим рекурсивного спуску, запускається рекурсивний аналізатор:

* Кожне правило граматики перевіряється на відповідність лексемам у поточній позиції.
* Якщо знайдено відповідність, індекс лексем збільшується, і аналіз продовжується для наступних правил.
* У разі помилки повертається інформація про невідповідність лексеми.

**4. Побудова абстрактного синтаксичного дерева (AST)**

* Дерево будується функцією buildAST. Кожен вузол представляє або термінальний, або нетермінальний символ.
* Для кожного правила створюються дочірні вузли, які відповідають його елементам.
* Якщо правило має два елементи справа, дерево будується рекурсивно для обох піддерев.

**5. Виведення AST**

Для візуалізації AST використовуються функції:

* printAST: виводить дерево в консоль у вигляді ієрархічної структури.
* printASTToFile: записує дерево у файл.

**6. Збереження таблиці CYK**

Таблиця результатів CYK може бути виведена або збережена у файл за допомогою функцій displayParseInfoTable та saveParseInfoTableToFile.

**7. Основна функція синтаксичного аналізу**

Функція syntaxAnalyze координує процес:

* Спочатку викликається метод CYK.
* Якщо CYK не успішний, виконується рекурсивний спуск.
* У разі помилки виводиться інформація про невідповідність та позицію помилки у вхідному коді.
  + 1. **Розробка дерева граматичного розбору.**

Program

├── "StartProgram"

├── "StartBlok"

├── {VariableDeclaration ";"}

│ ├── VariableDeclaration

│ │ ├── "Int32"

│ │ └── VariableList

│ │ ├── Identifier

│ │ │ ├── "\_"

│ │ │ └── LowUp {2}

│ │ │ ├── Up

│ │ │ │ └── "A" | "B" | ... | "Z"

│ │ │ ├── Low

│ │ │ │ └── "a" | "b" | ... | "z"

│ │ │ └── Digit

│ │ │ └── "0" | "1" | ... | "9"

│ │ └── {"," Identifier}

├── {Statement}

│ ├── Statement

│ │ ├── InputStatement

│ │ │ ├── "Scan"

│ │ │ └── Identifier

│ │ ├── OutputStatement

│ │ │ ├── "Print"

│ │ │ └── ArithmeticExpression

│ │ │ ├── LowPriorityExpression

│ │ │ │ ├── MiddlePriorityExpression

│ │ │ │ │ ├── Identifier

│ │ │ │ │ ├── Number

│ │ │ │ │ │ ├── ["-"]

│ │ │ │ │ │ └── Digit {5}

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

│ │ │ │ └── {MiddlePriorityOperator MiddlePriorityExpression}

│ │ │ └── {LowPriorityOperator LowPriorityExpression}

│ │ ├── AssignStatement

│ │ │ ├── ArithmeticExpression

│ │ │ └── "==>" Identifier

│ │ ├── IfElseStatement

│ │ │ ├── "If"

│ │ │ ├── "(" LogicalExpression ")"

│ │ │ │ ├── AndExpression

│ │ │ │ │ ├── Comparison

│ │ │ │ │ │ ├── ComparisonExpression

│ │ │ │ │ │ │ ├── ArithmeticExpression

│ │ │ │ │ │ │ ├── ComparisonOperator

│ │ │ │ │ │ │ └── ArithmeticExpression

│ │ │ │ │ │ └── [NotOperator] "(" LogicalExpression ")"

│ │ │ │ │ └── {AndOperator AndExpression}

│ │ │ │ └── {OrOperator 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

│ │ ├── "StartBlok"

│ │ ├── {Statement}

│ │ └── "EndBlok"

└── "EndBlok"

*Рис. 3.2. Дерево граматичного розбору.*

* + 1. **Розробка алгоритму роботи синтаксичного і семантичного аналізатора.**

На етапі семантичного аналізу необхідно виконати ідентифікацію ідентифікаторів, що складається з двох основних етапів:

* **Оголошення ідентифікаторів**: обробка інформації про нові ідентифікатори.
* **Використання ідентифікаторів**: перевірка коректності використання ідентифікаторів у коді.

Коли лексичний аналізатор знаходить ідентифікатор, він створює структуру, яка містить атрибути лексеми, зокрема її ім'я, тип і лексичний клас. Ця інформація передається семантичному аналізатору.

Якщо аналізується **оголошення ідентифікатора**, основним завданням семантичного аналізатора є додавання даних про ідентифікатор до таблиці ідентифікаторів.

Під час **використання ідентифікатора**, інформація з таблиці ідентифікаторів перевіряється на відповідність. Для визначення типу ідентифікатора достатньо звернутися до відповідного поля у цій таблиці.

* + 1. **Опис програми реалізації синтаксичного та семантичного аналізатора.**

Одним з найбільш простих і найбільш популярних методів низхідного синтаксичного аналізу є метод рекурсивного спуску (recursive descent method).

Метод заснований на тому, що в склад синтаксичного аналізатора входить множина рекурсивних процедур граматичного розбору, по одній для кожного правила граматики.



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

Семантичний аналіз і алгоритм ідентифікації ідентифікаторів

На етапі семантичного аналізу вирішується задача ідентифікації ідентифікаторів, що складається з двох частин:

* **Обробка оголошень ідентифікаторів:** додається інформація про ідентифікатори в таблицю ідентифікаторів.
* **Обробка використання ідентифікаторів:** перевіряється відповідність використання ідентифікаторів за інформацією з таблиці.

Лексичний аналізатор формує структуру, яка містить атрибути виділеної лексеми (ім’я, тип, лексичний клас) і передає її семантичному аналізатору. Семантична дія на цьому етапі залежить від того, чи йде мова про оголошення чи використання ідентифікатора.

Опис програми семантичного аналізатора

Семантичний аналізатор перевіряє логіку програми та коректність її структур. У коді реалізовано кілька ключових функцій:

Основні функції семантичного аналізатора

1. **getLastDataSectionLexemIndex**
   * Визначає індекс останньої лексеми в секції даних.
   * Використовує парсер recursiveDescentParserRuleWithDebug для перевірки секції "program\_\_\_\_part1".
   * Якщо лексема знайдена, повертається її індекс, інакше — код помилки.
2. **checkingInternalCollisionInDeclarations**
   * Перевіряє колізії в деклараціях ідентифікаторів та міток:
     + **Колізія ідентифікатор/ідентифікатор:** виявляється, якщо ідентифікатор задекларовано кілька разів у межах однієї області.
     + **Колізія мітка/мітка:** виявляється при дублюванні міток.
     + **Колізія ідентифікатор/мітка:** виявляється, якщо одне ім’я використовується і для змінної, і для мітки.
   * У разі недекларованого ідентифікатора чи мітки генерується помилка.
3. **checkingVariableInitialization**
   * Перевіряє, чи всі змінні ініціалізовані перед використанням.
   * Аналізує код після секції даних, перевіряючи операції присвоєння, виклики функцій чи введення даних.
4. **checkingCollisionInDeclarationsByKeyWords**
   * Виявляє збіги між іменами ідентифікаторів і зарезервованими словами:
     + Використовує регулярні вирази для аналізу.
     + У разі збігу повертає помилку (наприклад, COLLISION\_IK\_STATE).
5. **semantixAnalyze**
   * Координує всі перевірки:
     + Виявляє колізії в деклараціях.
     + Аналізує ініціалізацію змінних.
     + Перевіряє збіги з ключовими словами.
   * У разі помилок повертає відповідні коди стану.

Основні аспекти реалізації

1. **Робота з лексемами та граматикою:**
   * Таблиця лексем (lexemInfoTable) та граматика (Grammar) використовуються для аналізу на основі попередніх етапів.
   * Типи лексем визначаються через поле tokenType.
2. **Перевірка колізій:**
   * Виявляються конфлікти, щоб уникнути неоднозначності в коді.
3. **Використання регулярних виразів:**
   * Перевірки з ключовими словами виконуються за допомогою std::regex.
4. **Обробка помилок:**
   * Помилки відображаються у консоль із деталями
5. **Коди стану:**
   * Функції повертають коди стану для сигналізації про успішність аналізу чи виявлені помилки.

Типовий процес роботи

1. **Виклик функції semantixAnalyze:**
   * Виконується перевірка декларацій і виявлення колізій.
   * Аналізується ініціалізація змінних.
   * Перевіряються збіги з ключовими словами.
2. **У разі помилок:**
   * Генерується відповідний код стану.
   * Виводиться детальна інформація про проблему.

Цей алгоритм забезпечує надійний семантичний аналіз програми, дозволяючи виявляти помилки в логіці та деклараціях ще до виконання коду.



*Рис. 3.4. Графічне представлення роботи семантичного аналізатора*

* 1. Розробка генератора коду.

Генерація вихідного коду передбачає спочатку перетворення програми у якесь проміжне представлення, а тоді вже генерацію з проміжного представлення у вихідний код. У якості проміжного представлення виберемо абстрактне синтаксичне дерево.

Абстрактне синтаксичне дерево (AST) — це структура даних, яка представляє синтаксичну структуру вихідного коду програми у вигляді дерева. AST використовується в компіляторах, інтерпретаторах та інструментах статичного аналізу для обробки коду.

AST представляє тільки важливу для аналізу і виконання інформацію, ігноруючи зайві деталі (наприклад, круглі дужки чи крапки з комою). Це спрощений, але точний опис логіки програми.

Вузли дерева представляють конструкції мови програмування (оператори, вирази, змінні, функції тощо). Гілки відповідають підконструкціям або елементам цих конструкцій.

Кожен вузол відповідає певному типу конструкції коду (наприклад, оператору додавання, виклику функції, оголошенню змінної).

AST є спрощеною версією синтаксичного дерева. Воно не включає зайві вузли, що відповідають елементам, які не впливають на логіку програми (наприклад, дужки чи крапки з комою).

* + 1. **Розробка алгоритму роботи генератора коду.**

Алгоритм роботи генератора коду не є складним, проте є неординарним, оскільки не використовується звичний підхід до генерації коду з використанням AST дерева розбору.

Стадії генерації коду:

1. Генерація заголовку, коду включень та дата секції, чим займаються такі функції відповідно makeTitle, makeDependenciesDeclaration, makeDataSection;
2. Допоки є наступна лексема, перевіряти її тип та генерувати відповідний код.
3. Генерація футеру



*Рис. 3.5. Блок-сема алгоритму роботи генератора коду.*

* + 1. **Опис програми реалізації генератора коду.**

Основні функції і макроси забезпечують різні етапи генерації коду: створення секцій даних, секцій коду, ініціалізації змінних і структурування команд. Давайте розглянемо основні компоненти і їх призначення:

1. **Макроси та константи**

* **MAX\_TEXT\_SIZE, MAX\_GENERATED\_TEXT\_SIZE**: Визначають максимальний розмір тексту та згенерованого коду.
* **SUCCESS\_STATE**: Статус для успішного виконання.
* **MAX\_OUTTEXT\_SIZE**: Буфер для вихідного тексту.
* **MAX\_LEXEM\_SIZE**: Максимальний розмір однієї лексеми.
* **MAX\_WORD\_COUNT**: Максимальна кількість слів/лексем, які обробляються.

1. **Структури даних**

* **LabelOffsetInfo**:
  + Зберігає інформацію про мітки (label) та їх позиції в коді.
  + Використовується для управління стрибками (goto) в асемблерному коді.
* **GotoPositionInfo**:
  + Інформація про позиції інструкцій стрибків, які мають бути пов'язані з відповідними мітками.
* **tokenStruct**:
  + Таблиця, що описує багатокомпонентні токени, такі як IF ... THEN, FOR ... TO ..., WHILE, тощо.

1. **Генерація коду**

* **makeCode**:
  + Основна функція для генерації коду. Вона викликає кілька інших функцій для побудови різних секцій:
    - **makeTitle**: Генерує заголовок (наприклад, визначення моделі процесора та архітектури).
    - **makeDependenciesDeclaration**: Додає оголошення необхідних функцій і констант.
    - **makeDataSection**: Створює секцію даних.
    - **makeBeginProgramCode**: Починає секцію коду.
    - **makeInitCode**, **initMake**: Виконує ініціалізацію змінних.
    - **makeSaveHWStack**, **makeResetHWStack**: Зберігає та відновлює стек на апаратному рівні.
    - **makeEndProgramCode**: Додає фінальні інструкції (наприклад, ret для завершення програми).

1. **Маніпуляція з токенами**

* **detectMultiToken**:
  + Перевіряє, чи відповідає поточна лексема багатокомпонентному токену з таблиці tokenStruct.
* **createMultiToken**:
  + Створює багатокомпонентний токен і зберігає його у структурі LexemInfo.

1. **Генерація машинного коду**

* **outBytes2Code**:
  + Копіює байти з одного буфера до іншого, формуючи машинний код.
* **Пример генерації команд:**
  + **makeSaveHWStack**:
    - Генерує інструкцію mov ebp, esp для збереження стека.
  + **makeResetHWStack**:
    - Генерує інструкцію mov esp, ebp для відновлення стека.

**Як працює генерація коду в функції makeCode**

Функція makeCode поступово трансформує лексеми з таблиці LexemInfo у машинний код або інший низькорівневий формат. У цьому поясненні з кодовими вставками розглянемо, як саме це реалізовано.

1. **Ініціалізація**

На початку функція викликає кілька підфункцій для створення основних секцій коду:

currBytePtr = makeTitle(lastLexemInfoInTable, currBytePtr);

currBytePtr = makeDependenciesDeclaration(lastLexemInfoInTable, currBytePtr);

currBytePtr = makeDataSection(lastLexemInfoInTable, currBytePtr);

currBytePtr = makeBeginProgramCode(lastLexemInfoInTable, currBytePtr);

* **makeTitle**: Генерує заголовок програми
* **makeDependenciesDeclaration**: Додає секцію залежностей (наприклад, бібліотеки або модулі).
* **makeDataSection**: Додає секцію даних (глобальні змінні, константи тощо).
* **makeBeginProgramCode**: Додає інструкції для ініціалізації, наприклад, налаштування стеку чи регістрів.

1. **Ініціалізація стеку**

Перед початком основної генерації коду функція скидає тимчасовий стек і генерує інструкції для ініціалізації:

lexemInfoTransformationTempStackSize = 0;

currBytePtr = makeInitCode(lastLexemInfoInTable, currBytePtr);

currBytePtr = initMake(lastLexemInfoInTable, currBytePtr);

currBytePtr = makeSaveHWStack(lastLexemInfoInTable, currBytePtr);

* **makeInitCode**: Генерує код для ініціалізації змінних.

1. **Обробка лексем у циклі**

Основна логіка генерації знаходиться в циклі for, де кожна лексема обробляється залежно від її типу:

for (struct LexemInfo\* lastLexemInfoInTable\_;

lastLexemInfoInTable\_ = \*lastLexemInfoInTable,

(\*lastLexemInfoInTable)->lexemStr[0] != '\0'; ) {

Цей цикл ітерує через таблицю лексем, поки не зустріне лексему з порожнім рядком (lexemStr[0] == '\0').

1. **Генерація коду для конструкцій**

В залежності від лексеми, викликаються функції-генератори. Наприклад:

**Умовні оператори:**

IF\_THEN\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

ELSE\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

* **IF\_THEN\_CODER**: Додає інструкції для умовного оператора if.
* **ELSE\_CODER**: Генерує код для гілки else.

**Цикли:**

FOR\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

WHILE\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

REPEAT\_UNTIL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

* **FOR\_CODER**: Генерує код для циклу for.
* **WHILE\_CODER**: Генерує інструкції для циклу while.
* **REPEAT\_UNTIL\_CODER**: Обробляє конструкцію циклу repeat until.

**Операції та оператори:**

ADD\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

SUB\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

MUL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

DIV\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

MOD\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

* Генерація арифметичних операцій (+, -, \*, /, %).

**Логічні оператори:**

AND\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

OR\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

NOT\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

* Логічні оператори &&, ||, !.

**Інші оператори:**

INPUT\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

OUTPUT\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

* **INPUT\_CODER**: Обробляє введення.
* **OUTPUT\_CODER**: Обробляє виведення.

**5. Обробка помилок**

Якщо лексема не була оброблена жодною з функцій-генераторів, генерується помилка:

if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) {

printf("\r\nError in the code generator! \"%s\" - unexpected token!\r\n", (\*lastLexemInfoInTable)->lexemStr);

exit(0);

}

Це простий механізм обробки помилок, який завершує програму з повідомленням про неочікувану лексему.

**6. Завершення програми**

Після обробки всіх лексем функція генерує завершальні інструкції:

currBytePtr = makeResetHWStack(lastLexemInfoInTable, currBytePtr);

currBytePtr = makeEndProgramCode(lastLexemInfoInTable, currBytePtr);

* **makeResetHWStack**: Відновлює стан стеку.
* **makeEndProgramCode**: Додає фінальні інструкції, наприклад, завершення виконання.

**7. Виведення коду**

Функція viewCode виводить згенерований код форматі (шістнадцяткові):

void viewCode(unsigned char\* outCodePtr, size\_t outCodePrintSize, unsigned char align) {

printf("\r\n; +0x0 +0x1 +0x2 +0x3 +0x4 +0x5 +0x6 +0x7 +0x8 +0x9 +0xA +0xB +0xC +0xD +0xE +0xF ");

printf("\r\n;0x00000000: ");

// Вивід кожного байта

1. Налагодження та тестування розробленого транслятора

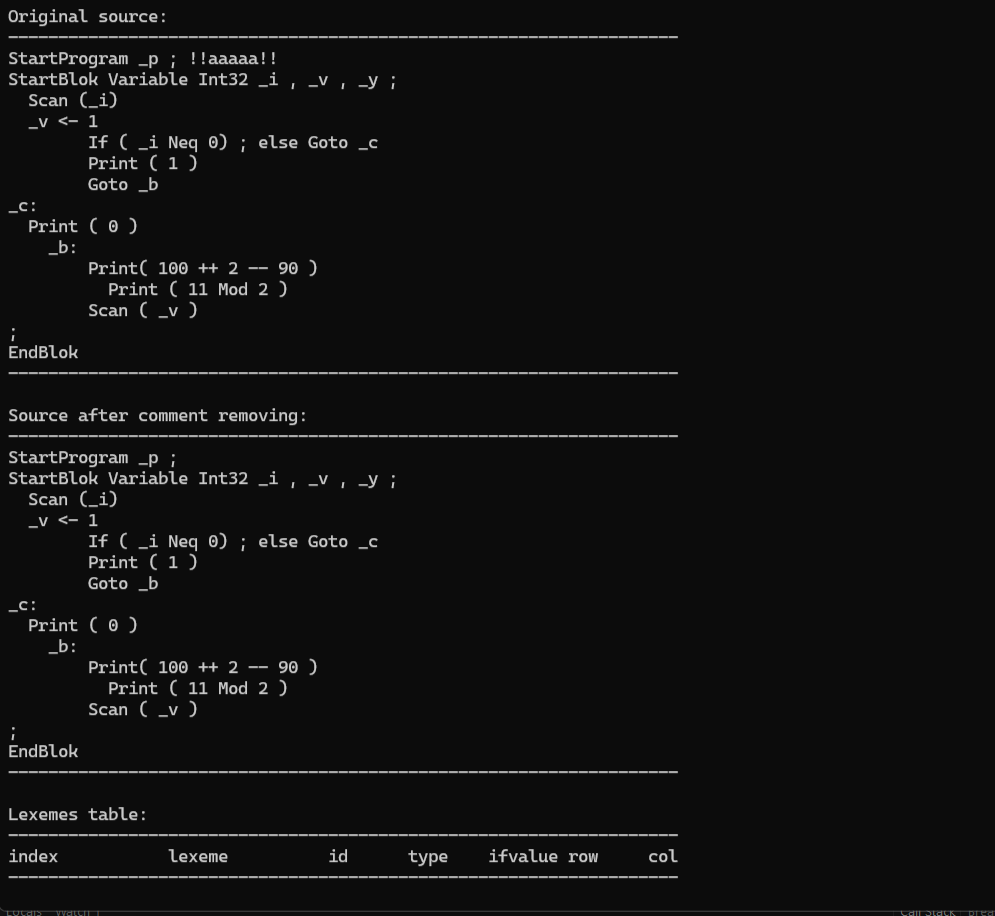
Будь-яке програмне забезпечення необхідно протестувати і налагодити. Після опрацювання синтаксичних і семантичних помилок необхідно переконатися, що розроблене програмне забезпечення функціонує так, як очікувалось.

Для перевірки коректності роботи розробленого транслятора необхідно буде написати тестові задачі на вхідній мові програмування, отримати код на мові програмування С і переконатись, що він працює правильно.

* 1. Опис інтерфейсу та інструкції користувачу.

Розроблений транслятор має простий консольний інтерфейс.

При запуску програми обирається файл file1.cwl:

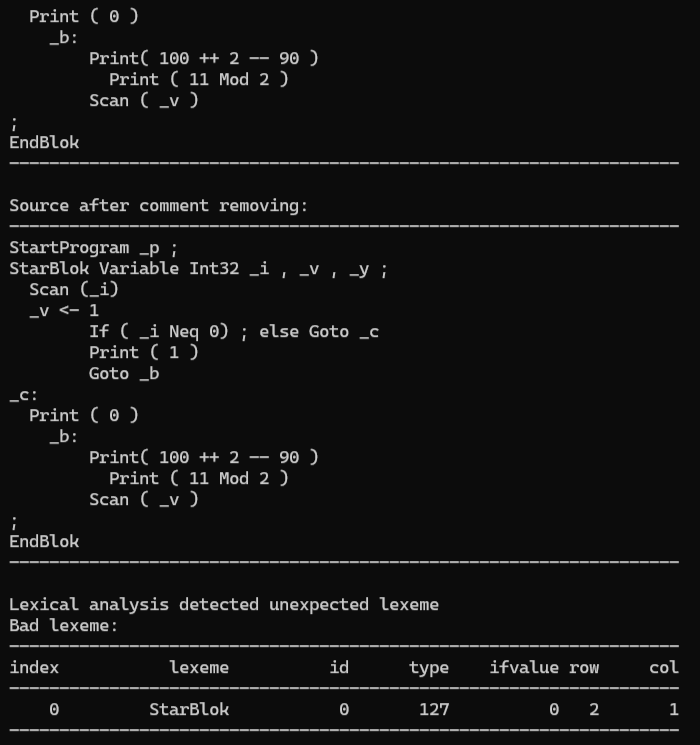


*Рис. 4.2.Результати роботи розробленого транслятора.*

* 1. Виявлення лексичних і синтаксичних помилок.

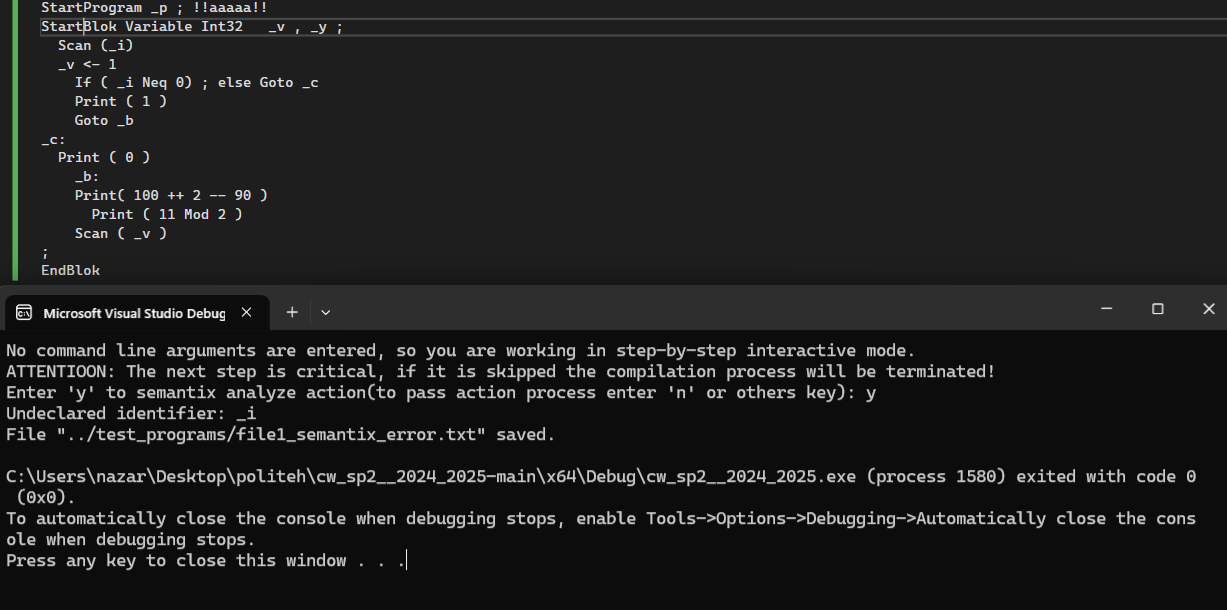
Помилки у вхідній програмі виявляються на етапі синтаксичного і семантичного аналізу.

Наприклад, у програмі зробимо синтаксичну помилку у другому рядку у слові «StartBlok»(StarBlok):



*Рис. 4.3. Вивід інформації про синтаксичну помилку.*

Зробимо семантичну помилку – не оголосимо змінну “\_i”:



*Рис. 4.4. Вивід інформації про семантичну помилку.*

* 1. Перевірка роботи транслятора за допомогою тестових задач.

**Тестова програма «*Лінійний алгоритм*»**

1. Ввести два числа А і В (імена змінних можуть бути іншими і мають відповідати правилам запису ідентифікаторів згідно індивідуального завдання).
2. Обрахувати значення виразу

Х = (А - В) \* 10 + (А + В) / 10

1. Вивести значення Х на екран.

Напишемо програму на вхідній мові програмування:

StartProgram \_p ;

StartBlok Variable Int32 \_a , \_b , \_x ;

Scan (\_a)

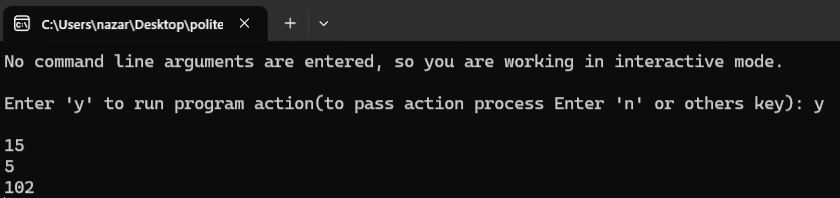
Scan (\_b)

\_x <- 10 \*\* (\_a -- \_b) ++ (\_a ++ \_b) Div 10

Print ( \_x )

Scan (\_a)

EndBlok



*Рис. 4.5. Результати виконання тестової задачі 1.*

**Тестова програма «*Алгоритм з розгалуженням*»**

1. Ввести три числа А, В, С (імена змінних можуть бути іншими і мають відповідати правилам запису ідентифікаторів згідно індивідуального завдання).

Використання вкладеного умовного оператора:

1. Знайти найбільше з них і вивести його на екран.

Використання простого умовного оператора:

1. Вивести на екран число 1, якщо усі числа однакові інакше вивести 0.

Напишемо програму на вхідній мові програмування:

StartProgram \_p ;

StartBlok Variable Int32 \_a , \_b , \_c ;

Scan (\_a)

Scan (\_b)

Scan (\_c)

If ( \_a Eq \_b ) ; else Goto \_r ;

Goto \_n

\_r :

Print ( 0 )

Goto \_k

\_n :

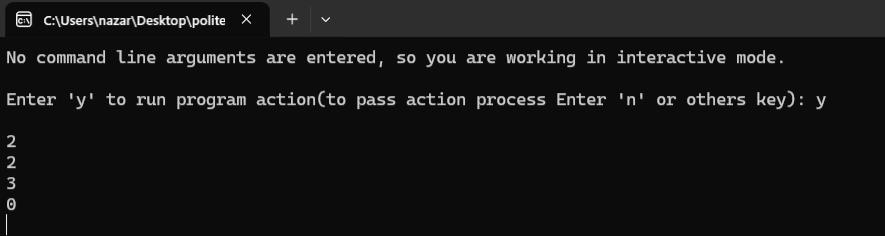
If ( \_a Eq \_c ) ; else Goto \_r ;

Print ( 1 )

\_k:

Scan (\_a)

EndBlok

****

*Рис. 4.6. Результати виконання тестової задачі 2.*

Висновки

У ході виконання курсового проєкту було реалізовано такі завдання:

1. Розроблено формальний опис мови програмування **z10** з використанням розширеної нотації Бекуса-Наура, визначено термінальні символи та ключові слова.
2. Створено наступні компоненти:  
   2.1. Лексичний аналізатор, призначений для розпізнавання лексем відповідно до формального опису мови.  
   2.2. Синтаксичний аналізатор, побудований на основі низхідного методу, із детальним описом вхідної мови в термінах розширеної нотації Бекуса-Наура.  
   2.3. Генератор коду, який викликає відповідні процедури після перевірки синтаксичним аналізатором правильності операторів мови програмування **p24**. Генерація коду реалізована мовою Assembler (x86).
3. Проведено тестування компілятора на тестових програмах за наступними аспектами:  
   3.1. Виявлення лексичних помилок.  
   3.2. Виявлення синтаксичних помилок.  
   3.3. Загальна перевірка функціональності компілятора.

У результаті виконання проєкту було освоєно методи розробки та реалізації основних компонентів систем програмування.

Список літературних джерел

1. Основи проектування трансляторів: Конспект лекцій : [Електронний ресурс]

: навч. посіб. для студ. спеціальності 123 – «Комп’ютерна інженерія» / О. І. Марченко ; КПІ ім. Ігоря Сікорського. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 108 с.

1. Формальні мови, граматики та автомати: Навчальний посібник / Гавриленко С.Ю. – Харків: НТУ «ХПІ», 2021. – 133 с.
2. Сопронюк Т.М. Системне програмування. Частина І. Елементи теорії формальних мов: Навчальний посібник у двох частинах. – Чернівці: ЧНУ, 2008. – 84 c.
3. Сопронюк Т.М. Системне програмування. Частина ІІ. Елементи теорії компіляції: Навчальний посібник у двох частинах. – Чернівці: ЧНУ, 2008. – 84 c.
4. Alfred V. Aho, Monica S. Lam, Ravi Seth, Jeffrey D. Ullma. Compilers, principles, techniques, and tools, Second Edition, New York, 2007. – 1038 c.
5. Системне програмування (курсовий проект) [Електронний ресурс] – Режим доступу до ресурсу: https://vns.lpnu.ua/course/view.php?id=11685.
6. 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 для тестових прикладів

Тестова програма «Лінійного алгоритму»

-------------------------------------------------------------------

Lexemes table:

-------------------------------------------------------------------

index lexeme id type ifvalue row col

-------------------------------------------------------------------

0 StartProgram 287 1 0 1 1

1 \_p 0 2 0 1 14

2 ; 256 1 0 1 17

3 StartBlok 309 1 0 2 1

4 Variable 300 1 0 2 11

5 Int32 426 1 0 2 20

6 \_a 1 2 0 2 26

7 , 270 1 0 2 29

8 \_b 2 2 0 2 31

9 , 270 1 0 2 29

10 \_x 3 2 0 2 36

11 ; 256 1 0 2 39

12 Scan 341 1 0 3 3

13 ( 282 1 0 3 8

14 \_a 1 2 0 3 9

15 ) 285 1 0 3 11

16 Scan 341 1 0 4 3

17 ( 282 1 0 4 8

18 \_b 2 2 0 4 9

19 ) 285 1 0 4 11

20 \_x 3 2 0 5 3

21 <- 258 1 0 5 6

22 10 320 4 10 5 9

23 \*\* 267 1 0 5 12

24 ( 282 1 0 5 16

25 \_a 1 2 0 5 17

26 -- 264 1 0 5 20

27 \_b 2 2 0 5 23

28 ) 285 1 0 5 25

29 ++ 261 1 0 5 27

30 ( 282 1 0 5 30

31 \_a 1 2 0 5 31

32 ++ 261 1 0 5 27

33 \_b 2 2 0 5 37

34 ) 285 1 0 5 39

35 Div 401 1 0 5 41

36 10 320 4 10 5 45

37 Print 346 1 0 6 2

38 ( 282 1 0 6 8

39 \_x 3 2 0 6 10

40 ) 285 1 0 6 13

41 Scan 341 1 0 7 3

42 ( 282 1 0 7 8

43 \_a 1 2 0 7 9

44 ) 285 1 0 7 11

45 EndBlok 319 1 0 8 1

-------------------------------------------------------------------

Тестова програма «Алгоритм з розгалуженням»

-------------------------------------------------------------------

Lexemes table:

-------------------------------------------------------------------

index lexeme id type ifvalue row col

-------------------------------------------------------------------

0 StartProgram 287 1 0 1 1

1 \_p 0 2 0 1 14

2 ; 256 1 0 1 17

3 StartBlok 309 1 0 2 1

4 Variable 300 1 0 2 11

5 Int32 426 1 0 2 20

6 \_a 1 2 0 2 26

7 , 270 1 0 2 29

8 \_b 2 2 0 2 31

9 , 270 1 0 2 29

10 \_c 3 2 0 2 36

11 ; 256 1 0 2 39

12 Scan 341 1 0 3 3

13 ( 282 1 0 3 8

14 \_a 1 2 0 3 9

15 ) 285 1 0 3 11

16 Scan 341 1 0 4 3

17 ( 282 1 0 4 8

18 \_b 2 2 0 4 9

19 ) 285 1 0 4 11

20 Scan 341 1 0 5 4

21 ( 282 1 0 5 9

22 \_c 3 2 0 5 10

23 ) 285 1 0 5 12

24 If 352 1 0 6 5

25 ( 282 1 0 6 8

26 \_a 1 2 0 6 10

27 Eq 272 1 0 6 13

28 \_b 2 2 0 6 16

29 ) 285 1 0 6 19

30 ; 256 1 0 6 21

31 else 355 1 0 6 23

32 Goto 396 1 0 6 28

33 \_r 4 2 0 6 33

34 ; 256 1 0 6 36

35 Goto 396 1 0 6 28

36 \_n 5 2 0 7 12

37 \_r 4 2 0 8 2

38 : 279 1 0 8 5

39 Print 346 1 0 9 3

40 ( 282 1 0 9 9

41 0 320 4 0 9 11

42 ) 285 1 0 9 13

43 Goto 396 1 0 10 3

44 \_k 6 2 0 10 8

45 \_n 5 2 0 11 4

46 : 279 1 0 11 7

47 If 352 1 0 12 6

48 ( 282 1 0 12 9

49 \_a 1 2 0 12 11

50 Eq 272 1 0 12 14

51 \_c 3 2 0 12 17

52 ) 285 1 0 12 20

53 ; 256 1 0 12 22

54 else 355 1 0 12 24

55 Goto 396 1 0 12 29

56 \_r 4 2 0 12 34

57 ; 256 1 0 12 22

58 Print 346 1 0 13 7

59 ( 282 1 0 13 13

60 1 320 4 1 13 15

61 ) 285 1 0 13 17

62 \_k 6 2 0 14 1

63 : 279 1 0 14 3

64 Scan 341 1 0 15 3

65 ( 282 1 0 15 8

66 \_a 1 2 0 15 9

67 ) 285 1 0 15 11

68 EndBlok 319 1 0 16 1

-------------------------------------------------------------------

Додаток Б. Код на асемблері, отриманий на виході транслятора для тестових прикладів;

Тестова програма «Лінійного алгоритму»

.686

.model flat, stdcall

option casemap : none

GetStdHandle proto STDCALL, nStdHandle : DWORD

ExitProcess proto STDCALL, uExitCode : DWORD

;MessageBoxA PROTO hwnd : DWORD, lpText : DWORD, lpCaption : DWORD, uType : DWORD

ReadConsoleA proto STDCALL, hConsoleInput : DWORD, lpBuffer : DWORD, nNumberOfCharsToRead : DWORD, lpNumberOfCharsRead : DWORD, lpReserved : DWORD

WriteConsoleA proto STDCALL, hConsoleOutput : DWORD, lpBuffert : DWORD, nNumberOfCharsToWrite : DWORD, lpNumberOfCharsWritten : DWORD, lpReserved : DWORD

wsprintfA PROTO C : VARARG

GetConsoleMode PROTO STDCALL, hConsoleHandle:DWORD, lpMode : DWORD

SetConsoleMode PROTO STDCALL, hConsoleHandle:DWORD, dwMode : DWORD

ENABLE\_LINE\_INPUT EQU 0002h

ENABLE\_ECHO\_INPUT EQU 0004h

.data

data\_start db 8192 dup (0)

;title\_msg db "Output:", 0

valueTemp\_msg db 256 dup(0)

valueTemp\_fmt db "%d", 10, 13, 0

;NumberOfCharsWritten dd 0

hConsoleInput dd 0

hConsoleOutput dd 0

buffer db 128 dup(0)

readOutCount dd ?

.code

start:

db 0E8h, 00h, 00h, 00h, 00h; call NexInstruction

;NexInstruction:

pop esi

sub esi, 5

mov edi, esi

add edi, 000004000h

mov ecx, edi

add ecx, 512

jmp initConsole

putProc PROC

push eax

push offset valueTemp\_fmt

push offset valueTemp\_msg

call wsprintfA

add esp, 12

;push 40h

;push offset title\_msg

;push offset valueTemp\_msg;

;push 0

;call MessageBoxA

push 0

push 0; offset NumberOfCharsWritten

push eax; NumberOfCharsToWrite

push offset valueTemp\_msg

push hConsoleOutput

call WriteConsoleA

ret

putProc ENDP

getProc PROC

push ebp

mov ebp, esp

push 0

push offset readOutCount

push 15

push offset buffer + 1

push hConsoleInput

call ReadConsoleA

lea esi, offset buffer

add esi, readOutCount

sub esi, 2

call string\_to\_int

mov esp, ebp

pop ebp

ret

getProc ENDP

string\_to\_int PROC

; input: ESI - string

; output: EAX - value

xor eax, eax

mov ebx, 1

xor ecx, ecx

convert\_loop :

movzx ecx, byte ptr[esi]

test ecx, ecx

jz done

sub ecx, '0'

imul ecx, ebx

add eax, ecx

imul ebx, ebx, 10

dec esi

jmp convert\_loop

done:

ret

string\_to\_int ENDP

initConsole:

push -10

call GetStdHandle

mov hConsoleInput, eax

push -11

call GetStdHandle

mov hConsoleOutput, eax

;push ecx

;push ebx

;push esi

;push edi

;push offset mode

;push hConsoleInput

;call GetConsoleMode

;mov ebx, eax

;or ebx, ENABLE\_LINE\_INPUT

;or ebx, ENABLE\_ECHO\_INPUT

;push ebx

;push hConsoleInput

;call SetConsoleMode

;pop edi

;pop esi

;pop ebx

;pop ecx

;hw stack save(save esp)

mov ebp, esp

;";"

;"4"

add ecx, 4

mov eax, 000000004h

mov dword ptr [ecx], eax

;"Scan"

mov eax, dword ptr[ecx]

mov edx, 000000044h

add edx, esi

push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

pop ecx

mov ebx, dword ptr[ecx]

sub ecx, 4

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"8"

add ecx, 4

mov eax, 000000008h

mov dword ptr [ecx], eax

;"Scan"

mov eax, dword ptr[ecx]

mov edx, 000000044h

add edx, esi

push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

pop ecx

mov ebx, dword ptr[ecx]

sub ecx, 4

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"12"

add ecx, 4

mov eax, 00000000Ch

mov dword ptr [ecx], eax

;"10"

add ecx, 4

mov eax, 00000000Ah

mov dword ptr [ecx], eax

;"\_a"

mov eax, edi

add eax, 000000004h

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"\_b"

mov eax, edi

add eax, 000000008h

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"--"

mov eax, dword ptr[ecx]

sub ecx, 4

sub dword ptr[ecx], eax

mov eax, dword ptr[ecx]

;"\*\*"

mov eax, dword ptr[ecx - 4]

;cdq

imul dword ptr [ecx]

sub ecx, 4

mov dword ptr [ecx], eax

;"\_a"

mov eax, edi

add eax, 000000004h

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"\_b"

mov eax, edi

add eax, 000000008h

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"++"

mov eax, dword ptr[ecx]

sub ecx, 4

add dword ptr[ecx], eax

mov eax, dword ptr[ecx]

;"10"

add ecx, 4

mov eax, 00000000Ah

mov dword ptr [ecx], eax

;"Div"

mov eax, dword ptr[ecx - 4]

cdq

idiv dword ptr [ecx]

sub ecx, 4

mov dword ptr [ecx], eax

;"++"

mov eax, dword ptr[ecx]

sub ecx, 4

add dword ptr[ecx], eax

mov eax, dword ptr[ecx]

;"<-"

mov eax, dword ptr[ecx]

mov ebx, dword ptr[ecx - 4]

sub ecx, 8

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"\_x"

mov eax, edi

add eax, 00000000Ch

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"Print"

mov eax, dword ptr[ecx]

mov edx, 00000001Bh

add edx, esi

;push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

;pop ecx

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"4"

add ecx, 4

mov eax, 000000004h

mov dword ptr [ecx], eax

;"Scan"

mov eax, dword ptr[ecx]

mov edx, 000000044h

add edx, esi

push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

pop ecx

mov ebx, dword ptr[ecx]

sub ecx, 4

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;hw stack reset(restore esp)

mov esp, ebp

xor eax, eax

ret

end start

Тестова програма «Алгоритм з розгалуженням»

.686

.model flat, stdcall

option casemap : none

GetStdHandle proto STDCALL, nStdHandle : DWORD

ExitProcess proto STDCALL, uExitCode : DWORD

;MessageBoxA PROTO hwnd : DWORD, lpText : DWORD, lpCaption : DWORD, uType : DWORD

ReadConsoleA proto STDCALL, hConsoleInput : DWORD, lpBuffer : DWORD, nNumberOfCharsToRead : DWORD, lpNumberOfCharsRead : DWORD, lpReserved : DWORD

WriteConsoleA proto STDCALL, hConsoleOutput : DWORD, lpBuffert : DWORD, nNumberOfCharsToWrite : DWORD, lpNumberOfCharsWritten : DWORD, lpReserved : DWORD

wsprintfA PROTO C : VARARG

GetConsoleMode PROTO STDCALL, hConsoleHandle:DWORD, lpMode : DWORD

SetConsoleMode PROTO STDCALL, hConsoleHandle:DWORD, dwMode : DWORD

ENABLE\_LINE\_INPUT EQU 0002h

ENABLE\_ECHO\_INPUT EQU 0004h

.data

data\_start db 8192 dup (0)

;title\_msg db "Output:", 0

valueTemp\_msg db 256 dup(0)

valueTemp\_fmt db "%d", 10, 13, 0

;NumberOfCharsWritten dd 0

hConsoleInput dd 0

hConsoleOutput dd 0

buffer db 128 dup(0)

readOutCount dd ?

.code

start:

db 0E8h, 00h, 00h, 00h, 00h; call NexInstruction

;NexInstruction:

pop esi

sub esi, 5

mov edi, esi

add edi, 000004000h

mov ecx, edi

add ecx, 512

jmp initConsole

putProc PROC

push eax

push offset valueTemp\_fmt

push offset valueTemp\_msg

call wsprintfA

add esp, 12

;push 40h

;push offset title\_msg

;push offset valueTemp\_msg;

;push 0

;call MessageBoxA

push 0

push 0; offset NumberOfCharsWritten

push eax; NumberOfCharsToWrite

push offset valueTemp\_msg

push hConsoleOutput

call WriteConsoleA

ret

putProc ENDP

getProc PROC

push ebp

mov ebp, esp

push 0

push offset readOutCount

push 15

push offset buffer + 1

push hConsoleInput

call ReadConsoleA

lea esi, offset buffer

add esi, readOutCount

sub esi, 2

call string\_to\_int

mov esp, ebp

pop ebp

ret

getProc ENDP

string\_to\_int PROC

; input: ESI - string

; output: EAX - value

xor eax, eax

mov ebx, 1

xor ecx, ecx

convert\_loop :

movzx ecx, byte ptr[esi]

test ecx, ecx

jz done

sub ecx, '0'

imul ecx, ebx

add eax, ecx

imul ebx, ebx, 10

dec esi

jmp convert\_loop

done:

ret

string\_to\_int ENDP

initConsole:

push -10

call GetStdHandle

mov hConsoleInput, eax

push -11

call GetStdHandle

mov hConsoleOutput, eax

;push ecx

;push ebx

;push esi

;push edi

;push offset mode

;push hConsoleInput

;call GetConsoleMode

;mov ebx, eax

;or ebx, ENABLE\_LINE\_INPUT

;or ebx, ENABLE\_ECHO\_INPUT

;push ebx

;push hConsoleInput

;call SetConsoleMode

;pop edi

;pop esi

;pop ebx

;pop ecx

;hw stack save(save esp)

mov ebp, esp

;";"

;"4"

add ecx, 4

mov eax, 000000004h

mov dword ptr [ecx], eax

;"Scan"

mov eax, dword ptr[ecx]

mov edx, 000000044h

add edx, esi

push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

pop ecx

mov ebx, dword ptr[ecx]

sub ecx, 4

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"8"

add ecx, 4

mov eax, 000000008h

mov dword ptr [ecx], eax

;"Scan"

mov eax, dword ptr[ecx]

mov edx, 000000044h

add edx, esi

push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

pop ecx

mov ebx, dword ptr[ecx]

sub ecx, 4

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"12"

add ecx, 4

mov eax, 00000000Ch

mov dword ptr [ecx], eax

;"Scan"

mov eax, dword ptr[ecx]

mov edx, 000000044h

add edx, esi

push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

pop ecx

mov ebx, dword ptr[ecx]

sub ecx, 4

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"If"

;"\_a"

mov eax, edi

add eax, 000000004h

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"\_b"

mov eax, edi

add eax, 000000008h

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"Eq"

mov eax, dword ptr[ecx]

sub ecx, 4

cmp dword ptr[ecx], eax

sete al

and eax, 1

mov dword ptr[ecx], eax

;after cond expresion (after "If")

cmp eax, 0

jz LABEL@AFTER\_THEN\_00007FF79F9099B0

;";" (after "then"-part of If-operator)

mov eax, 1

LABEL@AFTER\_THEN\_00007FF79F9099B0:

;"else"

cmp eax, 0

jnz LABEL@AFTER\_ELSE\_00007FF79F90A628

;"Goto" previous ident "\_r"(as label)

jmp LABEL@000002448ADDB9B8

;null statement (non-context)

;";" (after "else")

LABEL@AFTER\_ELSE\_00007FF79F90A628:

;"Goto" previous ident "\_n"(as label)

jmp LABEL@000002448ADDDB38

;null statement (non-context)

;ident "\_r"(as label) previous ":"

LABEL@000002448ADDB9B8:

;"0"

add ecx, 4

mov eax, 000000000h

mov dword ptr [ecx], eax

;"Print"

mov eax, dword ptr[ecx]

mov edx, 00000001Bh

add edx, esi

;push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

;pop ecx

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;"Goto" previous ident "\_k"(as label)

jmp LABEL@000002448ADDDE08

;null statement (non-context)

;ident "\_n"(as label) previous ":"

LABEL@000002448ADDDB38:

;"If"

;"\_a"

mov eax, edi

add eax, 000000004h

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"\_c"

mov eax, edi

add eax, 00000000Ch

mov eax, dword ptr[eax]

add ecx, 4

mov dword ptr [ecx], eax

;"Eq"

mov eax, dword ptr[ecx]

sub ecx, 4

cmp dword ptr[ecx], eax

sete al

and eax, 1

mov dword ptr[ecx], eax

;after cond expresion (after "If")

cmp eax, 0

jz LABEL@AFTER\_THEN\_00007FF79F9122D8

;";" (after "then"-part of If-operator)

mov eax, 1

LABEL@AFTER\_THEN\_00007FF79F9122D8:

;"else"

cmp eax, 0

jnz LABEL@AFTER\_ELSE\_00007FF79F912F50

;"Goto" previous ident "\_r"(as label)

jmp LABEL@000002448ADDB9B8

;null statement (non-context)

;";" (after "else")

LABEL@AFTER\_ELSE\_00007FF79F912F50:

;"1"

add ecx, 4

mov eax, 000000001h

mov dword ptr [ecx], eax

;"Print"

mov eax, dword ptr[ecx]

mov edx, 00000001Bh

add edx, esi

;push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

;pop ecx

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;ident "\_k"(as label) previous ":"

LABEL@000002448ADDDE08:

;"4"

add ecx, 4

mov eax, 000000004h

mov dword ptr [ecx], eax

;"Scan"

mov eax, dword ptr[ecx]

mov edx, 000000044h

add edx, esi

push ecx

;push ebx

push esi

push edi

call edx

pop edi

pop esi

;pop ebx

pop ecx

mov ebx, dword ptr[ecx]

sub ecx, 4

add ebx, edi

mov dword ptr [ebx], eax

mov ecx, edi ; reset second stack

add ecx, 512 ; reset second stack

;null statement (non-context)

;hw stack reset(restore esp)

mov esp, ebp

xor eax, eax

ret

end start

Додаток В. Дерево граматичного розбору.



Додаток Г. Лістинг програми.

Add.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: add.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeAddCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_ADD);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_add\_stackTopByECX\_eax[] = { 0x01, 0x01 };

//const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_stackTopByECX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

}

else if(generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_ADD][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add dword ptr[ecx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_ADD][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] += opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

And.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: and.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeAndCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_AND);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_cmp\_eax\_0[] = { 0x83, 0xF8, 0x00 };

const unsigned char code\_\_setne\_al[] = { 0x0F, 0x95, 0xC0 };

const unsigned char code\_\_and\_eax\_1[] = { 0x83, 0xE0, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

//

const unsigned char code\_\_cmp\_stackTopByECX\_0[] = { 0x83, 0x39, 0x00 };

const unsigned char code\_\_setne\_dl[] = { 0x0F, 0x95, 0xC2 };

const unsigned char code\_\_and\_edx\_1[] = { 0x83, 0xE2, 0x01 };

//

const unsigned char code\_\_and\_eax\_edx[] = { 0x23, 0xC2 };

//

const unsigned char code\_\_mov\_stackTopByECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_eax\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_setne\_al, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_1, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

//

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_stackTopByECX\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_setne\_dl, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_edx\_1, 3);

//

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_edx, 2);

//

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_AND][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp eax, 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " setne al\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and eax, 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " cmp dword ptr[ecx], 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " setne dl\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and edx, 1\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " and eax, edx\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr[ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_AND][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] = opStack[opStackIndex - 1] && opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

cli.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: cw\_lex.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../../src/include/cli/cli.h"

#include "../../../src/include/def.h"

#include "../../../src/include/config.h"

#include "../../../src/include/generator/generator.h"

#include "../../../src/include/lexica/lexica.h"

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

unsigned long long int mode = 0;

char parameters[PARAMETERS\_COUNT][MAX\_PARAMETERS\_SIZE] = { "" };

void comandLineParser(int argc, char\* argv[], unsigned long long int\* mode, char(\*parameters)[MAX\_PARAMETERS\_SIZE]) {

char tempTemp[PATH\_NAME\_LENGH] = { '\0' }, \* tempPtrPrev, \* tempPtrNext, nameTemp[PATH\_NAME\_LENGH] = { '\0' };

char modesNotDefined = 1;

\*mode = 0;

for (int index = 1; index < argc; ++index) {

if (!strcmp(argv[index], "-lex")) {

\*mode |= LEXICAL\_ANALYZE\_MODE;

modesNotDefined = 0;

continue;

}

else if (!strcmp(argv[index], "-stx")) {

\*mode |= SYNTAX\_ANALYZE\_MODE;

modesNotDefined = 0;

continue;

}

else if (!strcmp(argv[index], "-smt")) {

\*mode |= SEMANTIX\_ANALYZE\_MODE;

modesNotDefined = 0;

continue;

}

else if (!strcmp(argv[index], "-gen")) {

\*mode |= MAKE\_ASSEMBLY | MAKE\_BINARY;

modesNotDefined = 0;

continue;

}

else if (!strcmp(argv[index], "-run")) {

\*mode |= RUN\_BINARY;

modesNotDefined = 0;

continue;

}

else if (!strcmp(argv[index], "-all")) {

\*mode |= LEXICAL\_ANALYZE\_MODE | SYNTAX\_ANALYZE\_MODE | SEMANTIX\_ANALYZE\_MODE | MAKE\_ASSEMBLY | MAKE\_BINARY | RUN\_BINARY;

modesNotDefined = 0;

continue;

}

else if (!strcmp(argv[index], "-d")) {

\*mode |= DEBUG\_MODE;

modesNotDefined = 0;

continue;

}

// other keys

// TODO:...

// input filename

strncpy(parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], argv[index], MAX\_PARAMETERS\_SIZE);

}

// default mode, if not entered manually

if (modesNotDefined) {

if (parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

\*mode = LEXICAL\_ANALYZE\_MODE | SYNTAX\_ANALYZE\_MODE | SEMANTIX\_ANALYZE\_MODE | MAKE\_ASSEMBLY | MAKE\_BINARY;

}

else {

\*mode = UNDEFINED\_MODE;// | INTERACTIVE\_MODE | ;

//printf("No command line arguments. Go to step-by-step interactive mode(1) or start instant processing by default(2)?\r\n\r\n");

}

}

if (\*mode & UNDEFINED\_MODE) {

\*mode |= INTERACTIVE\_MODE;

//\*mode |= DEFAULT\_MODE;

\*mode |= DEBUG\_MODE;

}

// default input filename, if not entered manually

if (parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

strcpy(parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], DEFAULT\_INPUT\_FILENAME);

//printf("Input filename not setted. Used defaule input filename \"%s\"\r\n\r\n", parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

char choice[2] = { parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0], parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][1] };

//std::cout << "Enter file name(Enter \"" << choice[0] << "\" to use default \"" DEFAULT\_INPUT\_FILE "\"):\n";

printf("Input filename not setted. Enter file name(or enter '%c' to use default \"%s\"): ", parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0], parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

//std::cin >> fileName;

(void)scanf("%s", parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER]/\*, MAX\_PARAMETERS\_SIZE\*/);

if (parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == choice[0] && parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][1] == '\0') {

parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][1] = choice[1];

}

printf("\r\n");

(void)getchar();

}

// pass INTERACTIVE\_MODE ?

if (\*mode & INTERACTIVE\_MODE) {

// if (\*mode & INTERACTIVE\_MODE) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

// }

printf("No command line arguments.\r\nGo to step-by-step interactive mode('y') or start instant processing by default('n' or other key): ");

// if (\*mode & INTERACTIVE\_MODE) {

if (getchar() == 'y') {

printf("Press Enter to next step");

(void)getchar();

(void)getchar();

}

else {

\*mode |= SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE;

}

// }

}

strncpy(nameTemp, parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], PATH\_NAME\_LENGH);

nameTemp[PATH\_NAME\_LENGH - 1] = '\0';

tempPtrPrev = nameTemp;

tempPtrNext = NULL;

for (; tempPtrNext = strstr(tempPtrPrev + 1, "."); tempPtrPrev = tempPtrNext);

if (tempPtrPrev != nameTemp) {

\*tempPtrPrev = '\0';

}

#if 0

strncpy(tempTemp, parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], PATH\_NAME\_LENGH);

tempPtrPrev = tempTemp;

tempPtrPrev[0] == '\"' ? ++tempPtrPrev : 0;

tempPtrNext = tempPtrPrev = strtok(tempPtrPrev, " .\\/:");

while (tempPtrNext != NULL) {

tempPtrNext = strtok(NULL, " .\\/:");

if (tempPtrPrev && tempPtrNext) {

strncpy(nameTemp, tempPtrPrev, PATH\_NAME\_LENGH);

}

tempPtrPrev = tempPtrNext;

}

#endif

// default temp filename, if not entered manually

if (\*mode & (MAKE\_LEXEMES\_SEQUENSE | INTERACTIVE\_MODE) && parameters[OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER], "\_lexemes.txt", PATH\_NAME\_LENGH - strlen(parameters[OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out lexemes sequense filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

//(void)getchar();

}

}

// default temp filename, if not entered manually

if (\*mode & (MAKE\_LEXEMES\_SEQUENSE | INTERACTIVE\_MODE) && parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], "\_lexeme\_error.txt", PATH\_NAME\_LENGH - strlen(parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out error lexeme filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

//(void)getchar();

}

}

// default temp filename, if not entered manually

if (\*mode & (MAKE\_AST | INTERACTIVE\_MODE) && parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER], ".ast", PATH\_NAME\_LENGH - strlen(parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out AST filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

//(void)getchar();

}

}

// default temp filename, if not entered manually

if (\*mode & (MAKE\_AST | INTERACTIVE\_MODE) && parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], "\_syntax\_error.txt", PATH\_NAME\_LENGH - strlen(parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out syntax error filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

//(void)getchar();

}

}

// default temp filename, if not entered manually

if (\*mode & ((MAKE\_C | MAKE\_ASSEMBLY | MAKE\_OBJECT | MAKE\_BINARY) | INTERACTIVE\_MODE) && parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], "\_semantix\_error.txt", PATH\_NAME\_LENGH - strlen(parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out semantix error filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

//(void)getchar();

}

}

// default temp filename, if not entered manually

if (\*mode & (MAKE\_LEXEMES\_SEQUENSE | INTERACTIVE\_MODE) && parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER], "\_prepared\_lexemes.txt", PATH\_NAME\_LENGH - strlen(parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out prepared lexemes sequense filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

//(void)getchar();

}

}

// default temp filename, if not entered manually

if (\*mode & (MAKE\_C | INTERACTIVE\_MODE) && parameters[OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER], ".c", PATH\_NAME\_LENGH - strlen(parameters[OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out C filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

//(void)getchar();

}

}

// default temp filename, if not entered manually

if ((\*mode & (MAKE\_ASSEMBLY | INTERACTIVE\_MODE)) && parameters[OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER], ".asm", PATH\_NAME\_LENGH - strlen(parameters[OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out assembly filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

}

}

// default input filename, if not entered manually

if (\*mode & (MAKE\_OBJECT | INTERACTIVE\_MODE) && parameters[OUT\_OBJECT\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_OBJECT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_OBJECT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], ".obj", PATH\_NAME\_LENGH - strlen(parameters[OUT\_OBJECT\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out object filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_OBJECT\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

}

}

// default input filename, if not entered manually

if (\*mode & (MAKE\_BINARY | INTERACTIVE\_MODE) && parameters[OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] == '\0') {

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

}

strncpy(parameters[OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER], nameTemp, PATH\_NAME\_LENGH);

strncat(parameters[OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER], ".exe", PATH\_NAME\_LENGH - strlen(parameters[OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER]));

printf("Out binary filename not setted. Used defaule input filename \"%s\"\r\n", parameters[OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (\*mode & INTERACTIVE\_MODE && !(\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("Press Enter to next step");

(void)getchar();

}

}

return;

}

// after using this function use free(void \*) function to release text buffer

size\_t loadSource(char\*\* text, char\* fileName) {

if (!fileName) {

printf("No input file name\r\n");

return 0;

}

FILE\* file = fopen(fileName, "rb");

if (file == NULL) {

printf("File not loaded\r\n");

return 0;

}

fseek(file, 0, SEEK\_END);

long fileSize\_ = ftell(file);

if (fileSize\_ >= MAX\_TEXT\_SIZE) {

printf("the file(%ld bytes) is larger than %d bytes\r\n", fileSize\_, MAX\_TEXT\_SIZE);

fclose(file);

exit(2); // TODO: ...

//return 0;

}

size\_t fileSize = fileSize\_;

rewind(file);

if (!text) {

printf("Load source error\r\n");

return 0;

}

\*text = (char\*)malloc(sizeof(char) \* (fileSize + 1));

if (\*text == NULL) {

fputs("Memory error", stderr);

fclose(file);

exit(2); // TODO: ...

//return 0;

}

size\_t result = fread(\*text, sizeof(char), fileSize, file);

if (result != fileSize) {

fputs("Reading error", stderr);

fclose(file);

exit(3); // TODO: ...

//return 0;

}

(\*text)[fileSize] = '\0';

fclose(file);

return fileSize;

}  
  
cw.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS // for using sscanf in VS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: cw.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//#pragma comment(linker, "/STACK:516777216")

#include <windows.h>

//#include <winbase.h>

//#include <winuser.h>

//#include <shlobj.h>

//#include <shlwapi.h>

//#include <objbase.h>

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

//#include "conio.h"

//#include "locale.h"

#include <direct.h>

#include <fstream>

#include <iostream>

#include <algorithm>//

#include <iterator>

#include <regex>

#include "src/include/def.h"

#include "src/include/config.h"

#include "src/include/cli/cli.h"

//bool reSetDefaultInputFileName\_ =

//strcpy(parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], "../test\_programs/file1.cwl")

//!= NULL;

#include "src/include/lexica/lexica.h"

#include "src/include/syntax/syntax.h"

#include "src/include/semantix/semantix.h"

#include "src/include/preparer/preparer.h"

#include "src/include/generator/generator.h"

struct LexemInfo lexemesInfoTableTemp[MAX\_WORD\_COUNT]; // = { { "", 0, 0, 0 } };

struct LexemInfo\* lastLexemInfoInTableTemp = lexemesInfoTableTemp; // first for begin

unsigned char new\_code[8 \* 1024 \* 1024] = { '\0' }; //

unsigned char tempCodeBuffer[8 \* 1024 \* 1024] = { '\0' };

unsigned char outCodeBuffer[8 \* 1024 \* 1024] = { '\0' };

unsigned char errorMessagesPtrToLastBytePtr[8 \* 1024 \* 1024] = { '\0' };

int main(int argc, char\* argv[]) {

PostMessage(GetForegroundWindow(), WM\_INPUTLANGCHANGEREQUEST, 2, (UINT)LoadKeyboardLayoutA("00000409", KLF\_ACTIVATE));

char path[PATH\_NAME\_LENGH];

char temp[2 \* PATH\_NAME\_LENGH];

char productionOut[MAX\_TEXT\_SIZE] = { 0 };

comandLineParser(argc, argv, &mode, parameters);

char\* text;

size\_t sourceSize = loadSource(&text, parameters[INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

if (!sourceSize) {

printf("Empty source . . .");

printf("Press Enter to exit . . .");

getchar();

return 0;

}

if (!\_getcwd(path, PATH\_NAME\_LENGH))

{

printf("getcwd error ...\r\n");

printf("Press Enter to exit . . .");

return -1;

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("ATTENTIOON: The next step is critical, if it is skipped the compilation process will be terminated!\r\n");

printf("Enter 'y' to lexical analyze action(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & LEXICAL\_ANALYZE\_MODE) {

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

printf("Original source:\r\n");

printf("-------------------------------------------------------------------\r\n");

printf("%s\r\n", text);

printf("-------------------------------------------------------------------\r\n\r\n");

}

int commentRemoverResult = commentRemover(text, "!!", "!!");

if (commentRemoverResult) {

printf("Comment remover return %d\r\n", commentRemoverResult);

printf("Press Enter to exit . . .");

(void)getchar();

return 0;

}

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

printf("Source after comment removing:\r\n");

printf("-------------------------------------------------------------------\r\n");

printf("%s\r\n", text);

printf("-------------------------------------------------------------------\r\n\r\n");

}

struct LexemInfo ifBadLexemeInfo = tokenize(text, &lastLexemInfoInTable, identifierIdsTable, lexicalAnalyze);

if (ifBadLexemeInfo.tokenType == UNEXPEXTED\_LEXEME\_TYPE) {

UNEXPEXTED\_LEXEME\_TYPE;

ifBadLexemeInfo.tokenType;

printf("Lexical analysis detected unexpected lexeme\r\n");

printLexemes(&ifBadLexemeInfo, 1);

if (parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

printLexemesToFile(lexemesInfoTable, 1, parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

}

printf("Press Enter to exit . . .");

(void)getchar();

return 0;

}

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

printLexemes(lexemesInfoTable, 0);

if (parameters[OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

printLexemesToFile(lexemesInfoTable, 0, parameters[OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

}

if (parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], (unsigned char\*)"No error.", strlen("No error."));

}

}

else {

printf("Lexical analysis complete success\r\n");

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("ATTENTIOON: The next step is critical, if it is skipped the compilation process will be terminated!\r\n");

printf("Enter 'y' to syntax analyze action(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & SYNTAX\_ANALYZE\_MODE) {

errorMessagesPtrToLastBytePtr[0] = '\0';

unsigned char\* errorMessagesPtrToLastBytePtr\_ = errorMessagesPtrToLastBytePtr;

if (SUCCESS\_STATE != syntaxAnalyze(lexemesInfoTable, &grammar, DEFAULT\_SYNTAX\_ANAlYZE\_MODE, parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER], (char\*)&errorMessagesPtrToLastBytePtr)) { // TODO: add AST param

if (parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], errorMessagesPtrToLastBytePtr, strlen((const char\*)errorMessagesPtrToLastBytePtr));

}

if (parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER], (unsigned char\*)"AST build failed.", strlen("AST build failed."));

}

return 0;

}

if (parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], (unsigned char\*)"No error.", strlen("No error."));

printf("No error.\r\n");

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

else {

printf("\r\ncw terminated.");

return 0;

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("ATTENTIOON: The next step is critical, if it is skipped the compilation process will be terminated!\r\n");

printf("Enter 'y' to semantix analyze action(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & SEMANTIX\_ANALYZE\_MODE) {

errorMessagesPtrToLastBytePtr[0] = '\0';

unsigned char\* errorMessagesPtrToLastBytePtr\_ = errorMessagesPtrToLastBytePtr;

if (SUCCESS\_STATE != semantixAnalyze(lexemesInfoTable, &grammar, identifierIdsTable, (char \*)errorMessagesPtrToLastBytePtr)) {

if (parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], errorMessagesPtrToLastBytePtr, strlen((const char\*)errorMessagesPtrToLastBytePtr));

}

return 0;

}

if (parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER], (unsigned char\*)"No error.", strlen("No error."));

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

else {

printf("\r\ncw terminated.");

return 0;

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("ATTENTIOON: The next step is critical, if it is skipped the compilation process will be terminated!\r\n");

printf("Enter 'y' for the action to prepare for the compilation process(to pass the action process, enter 'n' or another key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & MAKE\_PREPARE) {

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

printLexemes(lexemesInfoTable, 0);

}

lastLexemInfoInTable = lexemesInfoTable;

makePrepare(lexemesInfoTable, &lastLexemInfoInTable, &lastLexemInfoInTableTemp);

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

printLexemes(lexemesInfoTableTemp, 0);

if (parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

printLexemesToFile(lexemesInfoTableTemp, 0, parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

}

}

else {

printf("Make prepare(expressions separation + creating reverse Polish notation) complete success\r\n");

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

if (parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

printf("File \"%s\" saved.\n", parameters[OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

}

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

else {

printf("\r\ncw terminated.");

return 0;

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("\r\n");

printf("Enter 'y' to create C-code(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & MAKE\_C) { // MAKE\_ASSEMBLY\_MODE

lastLexemInfoInTableTemp = lexemesInfoTableTemp;// printLexemes(lexemesInfoTableTemp, 0);

//outCodeBuffer[0] = '\0';

makeCode(&lastLexemInfoInTableTemp, outCodeBuffer, C\_CODER\_MODE); /\*The lexem info table will be changed and will need to be rebuilt. TODO: change the implementation!\*/

printf("\r\n\r\n%s\r\n\r\n", outCodeBuffer);

if (parameters[OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER], outCodeBuffer, strlen((const char\*)outCodeBuffer));

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("\r\n");

printf("Enter 'y' to create assembly(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & MAKE\_ASSEMBLY) { // MAKE\_ASSEMBLY\_MODE

lastLexemInfoInTableTemp = lexemesInfoTableTemp;// printLexemes(lexemesInfoTableTemp, 0);

lastLexemInfoInTable = lexemesInfoTable;

makePrepare(lexemesInfoTable, &lastLexemInfoInTable, &lastLexemInfoInTableTemp); /\* Rerun preapare process after previous etap. TODO: change the implementation!\*/

lastLexemInfoInTableTemp = lexemesInfoTableTemp;// printLexemes(lexemesInfoTableTemp, 0);

//outCodeBuffer[0] = '\0';

makeCode(&lastLexemInfoInTableTemp, outCodeBuffer, ASSEMBLY\_X86\_WIN32\_CODER\_MODE);

printf("\r\n\r\n%s\r\n\r\n", outCodeBuffer);

if (parameters[OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER][0] != '\0') {

writeBytesToFile(parameters[OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER], outCodeBuffer, strlen((const char\*)outCodeBuffer));

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

unsigned long long int byteCountWritedToTempCodeBuffer = 0;

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) { // BUILD NATIVE CODE

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("ATTENTIOON: The next step is critical, if it is skipped the compilation process will be terminated!\r\n");

printf("Enter 'y' to create native code(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & MAKE\_ASSEMBLY) {

lastLexemInfoInTableTemp = lexemesInfoTableTemp;// printLexemes(lexemesInfoTableTemp, 0);

lastLexemInfoInTable = lexemesInfoTable;

makePrepare(lexemesInfoTable, &lastLexemInfoInTable, &lastLexemInfoInTableTemp); /\* Rerun preapare process after previous etap. TODO: change the implementation!\*/

lastLexemInfoInTableTemp = lexemesInfoTableTemp;// printLexemes(lexemesInfoTableTemp, 0);

//outCodeBuffer[0] = '\0';

byteCountWritedToTempCodeBuffer = makeCode(&lastLexemInfoInTableTemp, tempCodeBuffer, MACHINE\_X86\_WIN32\_CODER\_MODE) - tempCodeBuffer;

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

viewCode(tempCodeBuffer, byteCountWritedToTempCodeBuffer, 16);

printf("\r\n\r\n");

}

else {

printf("Native code created complete successfully.\r\n\r\n");

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

else {

printf("\r\ncw terminated.");

return 0;

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("\r\n");

printf("Enter 'y' to create obj-file(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & MAKE\_OBJECT) {

unsigned long long int objectSize = buildTemplateForCodeObject(outCodeBuffer);

unsigned char\* currBytePtr = getObjectCodeBytePtr(outCodeBuffer, MACHINE\_X86\_WIN32\_CODER\_MODE);

(void)outBytes2Code(currBytePtr, tempCodeBuffer, byteCountWritedToTempCodeBuffer);

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

viewCode(outCodeBuffer, objectSize, 16);

printf("\r\n\r\n");

}

else {

printf("obj-code created complete successfully.\r\n\r\n");

}

writeBytesToFile(parameters[OUT\_OBJECT\_FILENAME\_WITH\_EXTENSION\_PARAMETER], outCodeBuffer, objectSize);

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

else if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

(void)getchar();

}

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in step-by-step interactive mode.\r\n");

printf("\r\n");

printf("Enter 'y' to create exe-file(to pass action process enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || getchar() == 'y') || mode & MAKE\_BINARY) {

unsigned long long int imageSize = buildTemplateForCodeImage(outCodeBuffer);

unsigned char\* currBytePtr = getImageCodeBytePtr(outCodeBuffer, MACHINE\_X86\_WIN32\_CODER\_MODE);

(void)outBytes2Code(currBytePtr, tempCodeBuffer, byteCountWritedToTempCodeBuffer);

if (mode & (DEBUG\_MODE | INTERACTIVE\_MODE)) {

viewCode(outCodeBuffer, imageSize, 16);

printf("\r\n\r\n");

}

else {

printf("exe-code created complete successfully.\r\n\r\n");

}

writeBytesToFile(parameters[OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER], outCodeBuffer, imageSize);

if (mode & INTERACTIVE\_MODE && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)) {

printf("\r\nPress Enter to next step");

(void)getchar();

(void)getchar();

}

}

else {

printf("\r\ncw terminated.");

return 0;

}

if (mode & INTERACTIVE\_MODE/\* && !(mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE)\*/) {

if (mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE) {

(void)getchar();

}

system("CLS");

fflush(stdin);

fflush(stdout);

fflush(stderr);

printf("No command line arguments are entered, so you are working in interactive mode.\r\n");

printf("\r\n");

printf("Enter 'y' to run program action(to pass action process Enter 'n' or others key): ");

}

fflush(stdin);

if (mode & INTERACTIVE\_MODE && (/\*mode & SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE || \*/getchar() == 'y') || mode & RUN\_BINARY) {

printf("\r\n");

//sprintf(temp, "START /b /wait \"\" /D \"%s\\masm32p\" %s.exe", path, parameters[OUT\_BINARY\_FILENAME\_WITHOUT\_EXTENSION\_PARAMETER]);

snprintf(temp, MAX\_PARAMETERS\_SIZE, "START /b /wait \"\" %s", parameters[OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER]);

fflush(stdin);

system((char\*)temp);

fflush(stdin);

}

else if (mode ^ RUN\_BINARY) {

printf("\r\n");

}

printf("\r\n\r\nPress Enter to exit . . .");

(void)getchar();

return 0;

}

div.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: div.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeDivCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_DIV);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECXMinus4[] = { 0x8B, 0x41, 0xFC };

const unsigned char code\_\_cdq[] = { 0x99 };

const unsigned char code\_\_idiv\_stackTopByECX[] = { 0xF7, 0x39 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_mov\_toAddrFromECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECXMinus4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cdq, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_idiv\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_toAddrFromECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_DIV][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx - 4]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cdq\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " idiv dword ptr [ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_DIV][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] /= opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
else.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: else.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

#include "string.h"

unsigned char\* makeElseCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_ELSE);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_cmp\_eax\_0[] = { 0x83, 0xF8, 0x00 };

const unsigned char code\_\_jnz\_offset[] = { 0x0F, 0x85, 0x00, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_eax\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jnz\_offset, 6);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_ELSE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_ELSE][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned long long int)(currBytePtr - 4);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, " cmp eax, 0\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jnz LABEL@AFTER\_ELSE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " if (opTemp != 0) goto LABEL\_\_AFTER\_ELSE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makePostElseCode\_(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned int)(currBytePtr - (unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue - 4);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@AFTER\_ELSE\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_AFTER\_ELSE\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

return currBytePtr;

}

unsigned char\* makeSemicolonAfterElseCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) { // Or Ender!

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_SEMICOLON);

if (multitokenSize

&&

lexemInfoTransformationTempStackSize

&&

!strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_ELSE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0], tokenStruct[MULTI\_TOKEN\_ELSE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0], tokenStruct[MULTI\_TOKEN\_ELSE][0]);

}

currBytePtr = makePostElseCode\_(lastLexemInfoInTable, currBytePtr, generatorMode);

--lexemInfoTransformationTempStackSize;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
equal.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: equal.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeIsEqualCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_EQUAL);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_cmp\_stackTopByECX\_eax[] = { 0x39, 0x01 };

const unsigned char code\_\_sete\_al[] = { 0x0F, 0x94, 0xC0 };

const unsigned char code\_\_and\_eax\_1[] = { 0x83, 0xE0, 0x01 };

const unsigned char code\_\_mov\_stackTopByECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_stackTopByECX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sete\_al, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_1, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_EQUAL][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp dword ptr[ecx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sete al\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and eax, 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr[ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_EQUAL][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] = opStack[opStackIndex - 1] == opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
  
for.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: for.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

#include "string.h"

unsigned char\* makeForCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_FOR);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeToOrDowntoCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) { // TODO: add assemblyBytePtr

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_DOWNTO);

bool toMode = false;

if (!multitokenSize) {

toMode = !!(multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_TO));

}

if (multitokenSize

&&

lexemInfoTransformationTempStackSize

&&

!strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_FOR][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_dec\_addrFromEBX[] = { 0xFF, 0x0B }; // dec dword ptr [ebx] // init

const unsigned char code\_\_inc\_addrFromEBX[] = { 0xFF, 0x03 }; // inc dword ptr [ebx] // init

const unsigned char code\_\_push\_ebx[] = { 0x53 }; // push ebx

if (toMode) {

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_dec\_addrFromEBX, 2); // init

}

else {

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_inc\_addrFromEBX, 2); // init

}

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_ebx, 1);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

if (toMode) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_TO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

else {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_DOWNTO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

}

else if (generatorMode == C\_CODER\_MODE) {

if (toMode) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_TO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

else {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_DOWNTO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned long long int)currBytePtr;

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

if (toMode) {

currBytePtr += sprintf((char\*)currBytePtr, " dec dword ptr [ebx]\r\n"); // start from (index - 1)

}

else {

currBytePtr += sprintf((char\*)currBytePtr, " inc dword ptr [ebx]\r\n"); // start from (index + 1)

}

currBytePtr += sprintf((char\*)currBytePtr, " push ebx\r\n");

if (toMode) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@AFTER\_TO\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@AFTER\_DOWNTO\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

}

else if (generatorMode == C\_CODER\_MODE) {

if (toMode) {

currBytePtr += sprintf((char\*)currBytePtr, " --data[lastBindDataIndex];\r\n"); // start from (index - 1)

}

else {

currBytePtr += sprintf((char\*)currBytePtr, " ++data[lastBindDataIndex];\r\n"); // start from (index + 1)

}

currBytePtr += sprintf((char\*)currBytePtr, " contextStack[++contextStackIndex] = lastBindDataIndex;\r\n");

if (toMode) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_AFTER\_TO\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_AFTER\_DOWNTO\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeDoCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_DO);

if (multitokenSize) {

bool toMode = false;

if (lexemInfoTransformationTempStackSize && !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_TO][0], MAX\_LEXEM\_SIZE)) {

toMode = true;

}

else if (lexemInfoTransformationTempStackSize < 2

|| strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_DOWNTO][0], MAX\_LEXEM\_SIZE)

|| strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_FOR][0], MAX\_LEXEM\_SIZE)

) {

return currBytePtr;

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_ebx\_addrFromESP[] = { 0x8B, 0x1C, 0x24 }; // mov ebx, dword ptr [esp]

const unsigned char code\_\_cmp\_addrFromEBX\_eax[] = { 0x39, 0x03 }; // cmp dword ptr [ebx], eax

const unsigned char code\_\_jge\_offset[] = { 0x0F, 0x8D, 0x00, 0x00, 0x00, 0x00 }; // jge ?? ?? ?? ??

const unsigned char code\_\_jle\_offset[] = { 0x0F, 0x8E, 0x00, 0x00, 0x00, 0x00 }; // jle ?? ?? ?? ??

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ebx\_addrFromESP, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_addrFromEBX\_eax, 2);

if (toMode) {

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jge\_offset, 6);

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned long long int)(currBytePtr - 4);

}

else {

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jle\_offset, 6);

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned long long int)(currBytePtr - 4);

}

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

if (toMode) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" (after \"%s\" after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_DO][0], tokenStruct[MULTI\_TOKEN\_TO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

else {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" (after \"%s\" after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_DO][0], tokenStruct[MULTI\_TOKEN\_DOWNTO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

}

else if (generatorMode == C\_CODER\_MODE) {

if (toMode) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" (after \"%s\" after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_DO][0], tokenStruct[MULTI\_TOKEN\_TO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

else {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" (after \"%s\" after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_DO][0], tokenStruct[MULTI\_TOKEN\_DOWNTO][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

}

if (toMode) {

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned long long int)(currBytePtr - 4);

}

else {

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned long long int)(currBytePtr - 4);

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_inc\_addrFromEBX[] = { 0xFF, 0x03 }; // inc dword ptr [ebx]

const unsigned char code\_\_dec\_addrFromEBX[] = { 0xFF, 0x0B }; // dec dword ptr [ebx]

if (toMode) {

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_inc\_addrFromEBX, 2);

}

else {

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_dec\_addrFromEBX, 2);

}

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, " mov ebx, dword ptr [esp]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp dword ptr [ebx], eax\r\n");

if (toMode) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jge LABEL@EXIT\_FOR\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

currBytePtr += sprintf((char\*)currBytePtr, " inc dword ptr [ebx]\r\n");

}

else {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jle LABEL@EXIT\_FOR\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

currBytePtr += sprintf((char\*)currBytePtr, " dec dword ptr [ebx]\r\n");

}

}

else if (generatorMode == C\_CODER\_MODE) {

if (toMode) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " if (opStack[contextStack[contextStackIndex]] >= opTemp) goto LABEL\_\_EXIT\_FOR\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

currBytePtr += sprintf((char\*)currBytePtr, " ++opStack[contextStack[contextStackIndex]];\r\n");

}

else {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " if (opStack[contextStack[contextStackIndex]] <= opTemp) goto LABEL\_\_EXIT\_FOR\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

currBytePtr += sprintf((char\*)currBytePtr, " --opStack[contextStack[contextStackIndex]];\r\n");

}

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makePostForCode\_(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode, bool toMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == C\_CODER\_MODE) {

//

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)((unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue - currBytePtr);

\*(unsigned int\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned int)(currBytePtr - (unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue - 4);

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_add\_esp\_4[] = { 0x83, 0xC4, 0x04 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_esp\_4, 3);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

if (toMode) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@AFTER\_TO\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@AFTER\_DOWNTO\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@EXIT\_FOR\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

currBytePtr += sprintf((char\*)currBytePtr, " add esp, 4; add esp, 8\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

if (toMode) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_AFTER\_TO\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_AFTER\_DOWNTO\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_EXIT\_FOR\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

currBytePtr += sprintf((char\*)currBytePtr, " --contextStackIndex;\r\n");

}

return currBytePtr;

}

unsigned char\* makeSemicolonAfterForCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) { // Or Ender!

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_SEMICOLON);

bool toMode = false;

if (multitokenSize

&&

lexemInfoTransformationTempStackSize > 1

&&

!strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_FOR][0], MAX\_LEXEM\_SIZE)

&& (

!strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_DOWNTO][0], MAX\_LEXEM\_SIZE)

||

(toMode = !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_TO][0], MAX\_LEXEM\_SIZE))

)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0], tokenStruct[MULTI\_TOKEN\_FOR][0]);

}

currBytePtr = makePostForCode\_(lastLexemInfoInTable, currBytePtr, generatorMode, toMode);

lexemInfoTransformationTempStackSize -= 2;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
generator.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

// TODO: CHANGE BY fRESET() TO END

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: generator.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//#define IDENTIFIER\_LEXEME\_TYPE 2

//#define VALUE\_LEXEME\_TYPE 4

//#define VALUE\_SIZE 4

#ifndef \_\_cplusplus

#define bool int

#define false 0

#define true 1

#endif

#include "../../../src/include/def.h"

#include "../../../src/include/config.h"

#include "../../../src/include/generator/generator.h"

#include "../../../src/include/lexica/lexica.h"

#include "../../../src/include/syntax/syntax.h"

#include "../../../src/include/semantix/semantix.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//#define DEBUG\_MODE\_BY\_ASSEMBLY

//#define C\_CODER\_MODE 0x01

//#define ASSEMBLY\_X86\_WIN32\_CODER\_MODE 0x02

//#define OBJECT\_X86\_WIN32\_CODER\_MODE 0x04

//#define MACHINE\_CODER\_MODE 0x08

//

//unsigned char generatorMode = MACHINE\_CODER\_MODE;

#define MAX\_TEXT\_SIZE 8192

#define MAX\_GENERATED\_TEXT\_SIZE (MAX\_TEXT\_SIZE \* 6)

#define GENERATED\_TEXT\_SIZE\_ 32768

//#define GENERATED\_TEXT\_SIZE (MAX\_TEXT\_SIZE % MAX\_GENERATED\_TEXT\_SIZE) // ?

#define SUCCESS\_STATE 0

#define MAX\_OUTTEXT\_SIZE (8\*8192\*1024)

unsigned char outText[MAX\_OUTTEXT\_SIZE] = ""; // !!!

#define MAX\_TEXT\_SIZE 8192

#define MAX\_WORD\_COUNT (MAX\_TEXT\_SIZE / 5)

#define MAX\_LEXEM\_SIZE 1024

#if 0

#define CODEGEN\_DATA\_TYPE int

#define START\_DATA\_OFFSET 512

#define OUT\_DATA\_OFFSET (START\_DATA\_OFFSET + 512)

#define M1 1024

#define M2 1024

//unsigned long long int dataOffsetMinusCodeOffset = 0x00003000;

unsigned long long int dataOffsetMinusCodeOffset = 0x00004000;

//unsigned long long int codeOffset = 0x000004AF;

//unsigned long long int baseOperationOffset = codeOffset + 49;// 0x00000031;

unsigned long long int baseOperationOffset = 0x000004AF;

unsigned long long int putProcOffset = 0x0000001B;

unsigned long long int getProcOffset = 0x00000044;

//unsigned long long int startCodeSize = 64 - 14; // 50 // -1

#endif

struct LabelOffsetInfo {

char labelStr[MAX\_LEXEM\_SIZE];

unsigned char\* labelBytePtr;

// TODO: ...

};

struct LabelOffsetInfo labelsOffsetInfoTable[MAX\_WORD\_COUNT] = { { "", NULL/\*, 0, 0\*/ } };

struct LabelOffsetInfo\* lastLabelOffsetInfoInTable = labelsOffsetInfoTable; // first for begin

struct GotoPositionInfo { // TODO: by Index

char labelStr[MAX\_LEXEM\_SIZE];

unsigned char\* gotoInstructionPositionPtr;

// TODO: ...

};

struct GotoPositionInfo gotoPositionsInfoTable[MAX\_WORD\_COUNT] = { { "", NULL/\*, 0, 0\*/ } }; // TODO: by Index

struct GotoPositionInfo\* lastGotoPositionInfoInTable = gotoPositionsInfoTable; // first for begin

////////////////////////////////

//#include "src/include/generator/generator.h"

//unsigned char generatorMode = MACHINE\_CODER\_MODE;

char\* tokenStruct[MAX\_TOKEN\_STRUCT\_ELEMENT\_COUNT][MAX\_TOKEN\_STRUCT\_ELEMENT\_PART\_COUNT] = { NULL };

#if 0

static void intitTokenStruct\_\_OLD() {

//SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, MULTI\_TOKEN\_BITWISE\_NOT, ("~"), (""), (""), (""))

//

// a12345\_ptr = a12345;

//

tokenStruct[MULTI\_TOKEN\_BITWISE\_NOT][0] = (char\*)"~";

tokenStruct[MULTI\_TOKEN\_BITWISE\_AND][0] = (char\*)"&";

tokenStruct[MULTI\_TOKEN\_BITWISE\_OR][0] = (char\*)"|";

tokenStruct[MULTI\_TOKEN\_NOT][0] = (char\*)"NOT";

tokenStruct[MULTI\_TOKEN\_AND][0] = (char\*)"AND";

tokenStruct[MULTI\_TOKEN\_OR][0] = (char\*)"OR";

tokenStruct[MULTI\_TOKEN\_EQUAL][0] = (char\*)"==";

tokenStruct[MULTI\_TOKEN\_NOT\_EQUAL][0] = (char\*)"!=";

tokenStruct[MULTI\_TOKEN\_LESS][0] = (char\*)"<";

tokenStruct[MULTI\_TOKEN\_GREATER][0] = (char\*)">";

tokenStruct[MULTI\_TOKEN\_LESS\_OR\_EQUAL][0] = (char\*)"<=";

tokenStruct[MULTI\_TOKEN\_GREATER\_OR\_EQUAL][0] = (char\*)">=";

tokenStruct[MULTI\_TOKEN\_ADD][0] = (char\*)"+";

tokenStruct[MULTI\_TOKEN\_SUB][0] = (char\*)"-";

tokenStruct[MULTI\_TOKEN\_MUL][0] = (char\*)"\*";

tokenStruct[MULTI\_TOKEN\_DIV][0] = (char\*)"DIV";

tokenStruct[MULTI\_TOKEN\_MOD][0] = (char\*)"MOD";

tokenStruct[MULTI\_TOKEN\_BIND\_RIGHT\_TO\_LEFT][0] = (char\*)"<<";

tokenStruct[MULTI\_TOKEN\_BIND\_LEFT\_TO\_RIGHT][0] = (char\*)">>";

tokenStruct[MULTI\_TOKEN\_COLON][0] = (char\*)":";

tokenStruct[MULTI\_TOKEN\_GOTO][0] = (char\*)"GOTO";

tokenStruct[MULTI\_TOKEN\_IF][0] = (char\*)"IF"; tokenStruct[MULTI\_TOKEN\_IF][1] = (char\*)"(";

// tokenStruct[MULTI\_TOKEN\_IF\_][0] = (char\*)"IF"; // don't change this!

tokenStruct[MULTI\_TOKEN\_THEN][0] = (char\*)")";

// tokenStruct[MULTI\_TOKEN\_THEN\_][0] = (char\*)"NULL"; tokenStruct[MULTI\_TOKEN\_IF][1] = (char\*)"STATEMENT"; // don't change this!

tokenStruct[MULTI\_TOKEN\_ELSE][0] = (char\*)"ELSE";

tokenStruct[MULTI\_TOKEN\_FOR][0] = (char\*)"FOR";

tokenStruct[MULTI\_TOKEN\_TO][0] = (char\*)"TO";

tokenStruct[MULTI\_TOKEN\_DOWNTO][0] = (char\*)"DOWNTO";

tokenStruct[MULTI\_TOKEN\_DO][0] = (char\*)"DO"; // tokenStruct[MULTI\_TOKEN\_DO][1] = (char\*)":";

//

tokenStruct[MULTI\_TOKEN\_WHILE][0] = (char\*)"WHILE";

tokenStruct[MULTI\_TOKEN\_CONTINUE\_WHILE][0] = (char\*)"CONTINUE"; tokenStruct[MULTI\_TOKEN\_CONTINUE\_WHILE][1] = (char\*)"WHILE";

tokenStruct[MULTI\_TOKEN\_EXIT\_WHILE][0] = (char\*)"EXIT"; tokenStruct[MULTI\_TOKEN\_EXIT\_WHILE][1] = (char\*)"WHILE";

tokenStruct[MULTI\_TOKEN\_END\_WHILE][0] = (char\*)"END"; tokenStruct[MULTI\_TOKEN\_END\_WHILE][1] = (char\*)"WHILE";

//

//

tokenStruct[MULTI\_TOKEN\_REPEAT][0] = (char\*)"REPEAT";

tokenStruct[MULTI\_TOKEN\_UNTIL][0] = (char\*)"UNTIL";

//

//

tokenStruct[MULTI\_TOKEN\_INPUT][0] = (char\*)"GET";

tokenStruct[MULTI\_TOKEN\_OUTPUT][0] = (char\*)"PUT";

//

//

tokenStruct[MULTI\_TOKEN\_RLBIND][0] = (char\*)"<<";

tokenStruct[MULTI\_TOKEN\_LRBIND][0] = (char\*)">>";

//

tokenStruct[MULTI\_TOKEN\_SEMICOLON][0] = (char\*)";";

tokenStruct[MULTI\_TOKEN\_BEGIN][0] = (char\*)"BEGIN";

tokenStruct[MULTI\_TOKEN\_END][0] = (char\*)"END";

tokenStruct[MULTI\_TOKEN\_NULL\_STATEMENT][0] = (char\*)"NULL"; tokenStruct[MULTI\_TOKEN\_NULL\_STATEMENT][1] = (char\*)"STATEMENT";

// NULL\_STATEMENT null\_statement

// null statement

//return 0;

}

//char intitTokenStruct\_ = (intitTokenStruct\_\_OLD(), 0);

#endif

INIT\_TOKEN\_STRUCT\_NAME(0);

unsigned char detectMultiToken(struct LexemInfo\* lexemInfoTable, enum TokenStructName tokenStructName) {

if (lexemInfoTable == NULL) {

return false;

}

if (!strncmp(lexemInfoTable[0].lexemStr, tokenStruct[tokenStructName][0], MAX\_LEXEM\_SIZE)

&& (tokenStruct[tokenStructName][1] == NULL || tokenStruct[tokenStructName][1][0] == '\0' || !strncmp(lexemInfoTable[1].lexemStr, tokenStruct[tokenStructName][1], MAX\_LEXEM\_SIZE))

&& (tokenStruct[tokenStructName][2] == NULL || tokenStruct[tokenStructName][2][0] == '\0' || !strncmp(lexemInfoTable[2].lexemStr, tokenStruct[tokenStructName][2], MAX\_LEXEM\_SIZE))

&& (tokenStruct[tokenStructName][3] == NULL || tokenStruct[tokenStructName][3][0] == '\0' || !strncmp(lexemInfoTable[3].lexemStr, tokenStruct[tokenStructName][3], MAX\_LEXEM\_SIZE))) {

return !!(tokenStruct[tokenStructName][0] != NULL && tokenStruct[tokenStructName][0][0] != '\0')

+ !!(tokenStruct[tokenStructName][1] != NULL && tokenStruct[tokenStructName][1][0] != '\0')

+ !!(tokenStruct[tokenStructName][2] != NULL && tokenStruct[tokenStructName][2][0] != '\0')

+ !!(tokenStruct[tokenStructName][3] != NULL && tokenStruct[tokenStructName][3][0] != '\0')

;

}

else {

return 0;

}

}

unsigned char createMultiToken(struct LexemInfo\*\* lexemInfoTable, enum TokenStructName tokenStructName) {

if (lexemInfoTable == NULL || \*lexemInfoTable == NULL) {

return false;

}

if (tokenStruct[tokenStructName][0] != NULL && tokenStruct[tokenStructName][0][0] != '\0') {

strncpy(lexemInfoTable[0][0].lexemStr, tokenStruct[tokenStructName][0], MAX\_LEXEM\_SIZE);

lexemInfoTable[0][0].lexemId = 0;

lexemInfoTable[0][0].tokenType = 0;

lexemInfoTable[0][0].ifvalue = 0;

lexemInfoTable[0][0].row = ~0;

lexemInfoTable[0][0].col = ~0;

++\* lexemInfoTable;

}

else {

return 0;

}

if (tokenStruct[tokenStructName][1] != NULL && tokenStruct[tokenStructName][1][0] != '\0') {

strncpy((\*lexemInfoTable)->lexemStr, tokenStruct[tokenStructName][1], MAX\_LEXEM\_SIZE);

lexemInfoTable[0][0].lexemId = 0;

lexemInfoTable[0][0].tokenType = 0;

lexemInfoTable[0][0].ifvalue = 0;

lexemInfoTable[0][0].row = ~0;

lexemInfoTable[0][0].col = ~0;

++\* lexemInfoTable;

}

else {

return 1;

}

if (tokenStruct[tokenStructName][2] != NULL && tokenStruct[tokenStructName][2][0] != '\0') {

strncpy((\*lexemInfoTable)->lexemStr, tokenStruct[tokenStructName][2], MAX\_LEXEM\_SIZE);

lexemInfoTable[0][0].lexemId = 0;

lexemInfoTable[0][0].tokenType = 0;

lexemInfoTable[0][0].ifvalue = 0;

lexemInfoTable[0][0].row = ~0;

lexemInfoTable[0][0].col = ~0;

++\* lexemInfoTable;

}

else {

return 2;

}

if (tokenStruct[tokenStructName][3] != NULL && tokenStruct[tokenStructName][3][0] != '\0') {

strncpy((\*lexemInfoTable)->lexemStr, tokenStruct[tokenStructName][3], MAX\_LEXEM\_SIZE);

lexemInfoTable[0][0].lexemId = 0;

lexemInfoTable[0][0].tokenType = 0;

lexemInfoTable[0][0].ifvalue = 0;

lexemInfoTable[0][0].row = ~0;

lexemInfoTable[0][0].col = ~0;

++\* lexemInfoTable;

}

else {

return 3;

}

return 4;

}

//#define MAX\_ACCESSORY\_STACK\_SIZE 128

struct NonContainedLexemInfo lexemInfoTransformationTempStack[MAX\_ACCESSORY\_STACK\_SIZE];

unsigned long long int lexemInfoTransformationTempStackSize = 0;

//

unsigned long long int getVariableOffset(char\* identifierStr) {

for (unsigned long long int index = 0; identifierIdsTable[index][0] != '\0'; ++index) {

if (!strncmp(identifierIdsTable[index], identifierStr, MAX\_LEXEM\_SIZE)) {

return START\_DATA\_OFFSET + sizeof(CODEGEN\_DATA\_TYPE) \* index;

}

}

return OUT\_DATA\_OFFSET;

}

unsigned char\* outBytes2Code(unsigned char\* currBytePtr, unsigned char\* fragmentFirstBytePtr, unsigned long long int bytesCout) {

for (; bytesCout--; \*currBytePtr++ = \*fragmentFirstBytePtr++);

return currBytePtr;

}

unsigned char\* makeEndProgramCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_xor\_eax\_eax[] = { 0x33, 0xC0 };

const unsigned char code\_\_ret[] = { 0xC3 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_xor\_eax\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_ret, 1);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, "imul ebp, 4\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, "add esp, ebp\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, "xor ebp, ebp;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " xor eax, eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ret\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "end start\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " return 0;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "}");

}

return currBytePtr;

}

unsigned char\* makeTitle(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, ".686\r\n");

currBytePtr += sprintf((char\*)currBytePtr, ".model flat, stdcall\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "option casemap : none\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "#define \_CRT\_SECURE\_NO\_WARNINGS\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, ".model flat, stdcall\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, "option casemap : none\r\n");

}

return currBytePtr;

}

unsigned char\* makeDependenciesDeclaration(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "GetStdHandle proto STDCALL, nStdHandle : DWORD\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "ExitProcess proto STDCALL, uExitCode : DWORD\r\n");

currBytePtr += sprintf((char\*)currBytePtr, ";MessageBoxA PROTO hwnd : DWORD, lpText : DWORD, lpCaption : DWORD, uType : DWORD\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "ReadConsoleA proto STDCALL, hConsoleInput : DWORD, lpBuffer : DWORD, nNumberOfCharsToRead : DWORD, lpNumberOfCharsRead : DWORD, lpReserved : DWORD\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "WriteConsoleA proto STDCALL, hConsoleOutput : DWORD, lpBuffert : DWORD, nNumberOfCharsToWrite : DWORD, lpNumberOfCharsWritten : DWORD, lpReserved : DWORD\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "wsprintfA PROTO C : VARARG\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "GetConsoleMode PROTO STDCALL, hConsoleHandle:DWORD, lpMode : DWORD\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "SetConsoleMode PROTO STDCALL, hConsoleHandle:DWORD, dwMode : DWORD\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "ENABLE\_LINE\_INPUT EQU 0002h\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "ENABLE\_ECHO\_INPUT EQU 0004h\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "#include \"stdio.h\"\r\n");

}

#ifdef DEBUG\_MODE\_BY\_ASSEMBLY

#endif

return currBytePtr;

}

unsigned char\* makeDataSection(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, ".data\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " data\_start db 8192 dup (0)\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;title\_msg db \"Output:\", 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " valueTemp\_msg db 256 dup(0)\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " valueTemp\_fmt db \"%%d\", 10, 13, 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;NumberOfCharsWritten dd 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " hConsoleInput dd 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " hConsoleOutput dd 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " buffer db 128 dup(0)\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " readOutCount dd ?\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "int data[8192] = {0};\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "int contextStack[8192] = {0}, contextStackIndex = 0;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "int opStack[8192] = {0}, opStackIndex = 0, opTemp = 0;;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "int lastBindDataIndex = 0;\r\n");

}

return currBytePtr;

}

unsigned char\* makeBeginProgramCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, ".code\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "start:\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "int main() {\r\n");

}

return currBytePtr;

}

unsigned char\* makeInitCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

// unsigned char code\_\_call\_NexInstructionLabel[] = { 0xE8, 0x00, 0x00, 0x00, 0x00 };

//

// unsigned char code\_\_pop\_esi[] = { 0x5E };

// unsigned char code\_\_sub\_esi\_5[] = { 0x83, 0xEE, 0x05 };

// unsigned char code\_\_mov\_edi\_esi[] = { 0x8B, 0xFE };

// unsigned char code\_\_add\_edi\_dataOffsetMinusCodeOffset[] = { 0xE8, 0xC7, 0x00, 0x00, 0x00, 0x00 };

// //unsigned char code\_\_xor\_ebp\_ebp[] = { 0x33, 0xED };

// unsigned char code\_\_mov\_ecx\_edi[] = { 0x8B, 0xCF };

// unsigned char code\_\_add\_ecx\_512[] = { 0x81, 0xC1, 0x00, 0x02, 0x00, 0x00 };

// unsigned char code\_\_jmp\_initConsole[] = { 0xEB, 0x7C };

//

// currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_call\_NexInstructionLabel, 5);

// currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_esi, 1);

// currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_esi\_5, 3);

// currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_edi\_esi, 2);

// currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_edi\_dataOffsetMinusCodeOffset, 6);

// \*(unsigned int \*)(currBytePtr - 4) = dataOffsetMinusCodeOffset;

// //currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_xor\_ebp\_ebp, 2);

// currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ecx\_edi, 2);

// currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ecx\_512, 6);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " db 0E8h, 00h, 00h, 00h, 00h; call NexInstruction\r\n");

currBytePtr += sprintf((char\*)currBytePtr, ";NexInstruction:\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " pop esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub esi, 5\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov edi, esi\r\n");//printf(" mov edi, offset data\_start\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add edi, 0%08Xh\r\n", (int)dataOffsetMinusCodeOffset);

//currBytePtr += sprintf((char\*)currBytePtr, " xor ebp, ebp\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ecx, edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ecx, 512\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " jmp initConsole\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " putProc PROC\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push offset valueTemp\_fmt\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push offset valueTemp\_msg\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call wsprintfA\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add esp, 12\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push 40h\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push offset title\_msg\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push offset valueTemp\_msg;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;call MessageBoxA\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push 0; offset NumberOfCharsWritten\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push eax; NumberOfCharsToWrite\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push offset valueTemp\_msg\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push hConsoleOutput\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call WriteConsoleA\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ret\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " putProc ENDP\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " getProc PROC\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " push eax\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " push offset valueTemp\_fmt\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " push offset valueTemp\_msg\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " call wsprintfA\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " add esp, 12\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " push 40h\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " push offset title\_msg\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " push offset valueTemp\_msg;\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " push 0\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " call MessageBoxA\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " ret\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " getProc ENDP\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " getProc PROC\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push ebp\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ebp, esp\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push offset readOutCount\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push 15\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push offset buffer + 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push hConsoleInput\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call ReadConsoleA\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " lea esi, offset buffer\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add esi, readOutCount\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub esi, 2\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call string\_to\_int\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov esp, ebp\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " pop ebp\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ret\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " getProc ENDP\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " string\_to\_int PROC\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ; input: ESI - string\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ; output: EAX - value\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " xor eax, eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ebx, 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " xor ecx, ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "convert\_loop :\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " movzx ecx, byte ptr[esi]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " test ecx, ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " jz done\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, '0'\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " imul ecx, ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add eax, ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " imul ebx, ebx, 10\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " dec esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " jmp convert\_loop\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "done:\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ret\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " string\_to\_int ENDP\r\n");

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " initConsole:\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push -10\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call GetStdHandle\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov hConsoleInput, eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push -11\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call GetStdHandle\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov hConsoleOutput, eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " \r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push offset mode\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push hConsoleInput\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;call GetConsoleMode\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;mov ebx, eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;or ebx, ENABLE\_LINE\_INPUT \r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;or ebx, ENABLE\_ECHO\_INPUT\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push hConsoleInput\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;call SetConsoleMode\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;pop edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;pop esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;pop ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;pop ecx\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

//currBytePtr += sprintf((char\*)currBytePtr, " int data[8192] = {0};\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " int contextStack[8192] = {0}, contextStackIndex = 0;\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " int opStack[8192] = {0}, opStackIndex = 0, opTemp = 0;;\r\n");

//currBytePtr += sprintf((char\*)currBytePtr, " int lastBindDataIndex = 0;\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " contextStackIndex = 0;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " opStackIndex = 0;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = 0;\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " lastBindDataIndex = 0;\r\n");

}

#ifdef DEBUG\_MODE\_BY\_ASSEMBLY

#endif

return currBytePtr;

}

//

#include "../../../src/include/preparer/preparer.h"

//

//

#include "../../../src/include/generator/bitwise\_not.h"

#include "../../../src/include/generator/bitwise\_and.h"

#include "../../../src/include/generator/bitwise\_or.h"

#include "../../../src/include/generator/not.h"

#include "../../../src/include/generator/and.h"

#include "../../../src/include/generator/or.h"

//

#include "../../../src/include/generator/add.h"

#include "../../../src/include/generator/sub.h"

#include "../../../src/include/generator/mul.h"

#include "../../../src/include/generator/div.h"

#include "../../../src/include/generator/mod.h"

//

#include "../../../src/include/generator/null\_statement.h"

#include "../../../src/include/generator/operand.h"

#include "../../../src/include/generator/input.h"

#include "../../../src/include/generator/output.h"

#include "../../../src/include/generator/equal.h"

#include "../../../src/include/generator/not\_equal.h"

#include "../../../src/include/generator/less.h"

#include "../../../src/include/generator/greater.h"

#include "../../../src/include/generator/less\_or\_equal.h"

#include "../../../src/include/generator/greater\_or\_equal.h"

#include "../../../src/include/generator/rlbind.h"

#include "../../../src/include/generator/lrbind.h"

#include "../../../src/include/generator/goto\_label.h"

#include "../../../src/include/generator/if\_then.h"

#include "../../../src/include/generator/else.h"

#include "../../../src/include/generator/for.h"

#include "../../../src/include/generator/while.h"

#include "../../../src/include/generator/repeat\_until.h"

//

#include "../../../src/include/generator/semicolon.h"

//

unsigned char\* initMake(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

//return currBytePtr;

unsigned long long int lastDataSectionLexemIndex = getDataSectionLastLexemIndex(\*lastLexemInfoInTable, &grammar);

if(lastDataSectionLexemIndex == ~0) {

printf("Error: bad section!\r\n");

exit(0);

}

labelInfoTable.clear(); /\* It's not pretty. TODO: change it. \*/

\*lastLexemInfoInTable += lastDataSectionLexemIndex;

return currBytePtr;

}

unsigned char\* makeSaveHWStack(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_ebp\_esp[] = { 0x8B, 0xEC };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ebp\_esp, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;hw stack save(save esp)\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ebp, esp\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

//

}

return currBytePtr;

}

unsigned char\* makeResetHWStack(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_esp\_ebp[] = { 0x8B, 0xE5 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_esp\_ebp, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;hw stack reset(restore esp)\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov esp, ebp\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

//

}

#ifdef DEBUG\_MODE\_BY\_ASSEMBLY

#endif

return currBytePtr;

}

unsigned char\* noMake(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr) {

if (!strncmp((\*lastLexemInfoInTable)->lexemStr, T\_NAME\_0, MAX\_LEXEM\_SIZE)

|| !strncmp((\*lastLexemInfoInTable)->lexemStr, T\_DATA\_0, MAX\_LEXEM\_SIZE)

|| !strncmp((\*lastLexemInfoInTable)->lexemStr, T\_BODY\_0, MAX\_LEXEM\_SIZE)

|| !strncmp((\*lastLexemInfoInTable)->lexemStr, T\_DATA\_TYPE\_0, MAX\_LEXEM\_SIZE)

|| !strncmp((\*lastLexemInfoInTable)->lexemStr, T\_COMA\_0, MAX\_LEXEM\_SIZE)

|| !strncmp((\*lastLexemInfoInTable)->lexemStr, T\_END\_0, MAX\_LEXEM\_SIZE)

) {

return ++ \* lastLexemInfoInTable, currBytePtr;

}

return currBytePtr;

}

unsigned char\* createPattern() {

return NULL;

}

unsigned char\* getObjectCodeBytePtr(unsigned char\* baseBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

}

else if (generatorMode == C\_CODER\_MODE) {

}

return baseBytePtr + baseOperationObjectOffset;

}

unsigned char\* getImageCodeBytePtr(unsigned char\* baseBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

}

else if (generatorMode == C\_CODER\_MODE) {

}

return baseBytePtr + baseOperationOffset;

}

unsigned char\* makeCode(struct LexemInfo\*\* lastLexemInfoInTable/\*TODO:...\*/, unsigned char\* currBytePtr, unsigned char generatorMode) { // TODO:...

currBytePtr = makeTitle(lastLexemInfoInTable, currBytePtr, generatorMode);

currBytePtr = makeDependenciesDeclaration(lastLexemInfoInTable, currBytePtr, generatorMode);

currBytePtr = makeDataSection(lastLexemInfoInTable, currBytePtr, generatorMode);

currBytePtr = makeBeginProgramCode(lastLexemInfoInTable, currBytePtr, generatorMode);

lexemInfoTransformationTempStackSize = 0;

currBytePtr = makeInitCode(lastLexemInfoInTable, currBytePtr, generatorMode);

currBytePtr = initMake(lastLexemInfoInTable, currBytePtr, generatorMode);

currBytePtr = makeSaveHWStack(lastLexemInfoInTable, currBytePtr, generatorMode);

for (struct LexemInfo\* lastLexemInfoInTable\_; lastLexemInfoInTable\_ = \*lastLexemInfoInTable, (\*lastLexemInfoInTable)->lexemStr[0] != '\0';) {

LABEL\_GOTO\_LABELE\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//

IF\_THEN\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

ELSE\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//

//currBytePtr = makeForCycleCode(lastLexemInfoInTable, currBytePtr);

//currBytePtr = makeToOrDowntoCycleCode(lastLexemInfoInTable, currBytePtr);

//currBytePtr = makeDoCycleCode(lastLexemInfoInTable, currBytePtr);

//currBytePtr = makeSemicolonAfterForCycleCode(lastLexemInfoInTable, currBytePtr);

FOR\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//

WHILE\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//

//

REPEAT\_UNTIL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeValueCode(lastLexemInfoInTable, currBytePtr);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeIdentifierCode(lastLexemInfoInTable, currBytePtr);

OPERAND\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeNotCode(lastLexemInfoInTable, currBytePtr);

BITWISE\_NOT\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

BITWISE\_AND\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

BITWISE\_OR\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

NOT\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

AND\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

OR\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

EQUAL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

NOT\_EQUAL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

LESS\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

GREATER\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

LESS\_OR\_EQUAL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

GREATER\_OR\_EQUAL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeAddCode(lastLexemInfoInTable, currBytePtr);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeSubCode(lastLexemInfoInTable, currBytePtr);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeMulCode(lastLexemInfoInTable, currBytePtr);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeDivCode(lastLexemInfoInTable, currBytePtr);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeModCode(lastLexemInfoInTable, currBytePtr);

ADD\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

SUB\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

MUL\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

DIV\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

MOD\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeRightToLeftBindCode(lastLexemInfoInTable, currBytePtr);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeLeftToRightBindCode(lastLexemInfoInTable, currBytePtr);

INPUT\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

OUTPUT\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeGetCode(lastLexemInfoInTable, currBytePtr);

//if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makePutCode(lastLexemInfoInTable, currBytePtr);

RLBIND\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

LRBIND\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

///\* (1) Ignore phase\*/if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeSemicolonAfterNonContextCode(lastLexemInfoInTable, currBytePtr);

///\* (2) Ignore phase\*/if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) currBytePtr = makeSemicolonIgnoreContextCode(lastLexemInfoInTable, currBytePtr);

NON\_CONTEXT\_SEMICOLON\_CODER(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

NON\_CONTEXT\_NULL\_STATEMENT(lastLexemInfoInTable\_, lastLexemInfoInTable, currBytePtr, generatorMode, NULL);

if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) {

currBytePtr = noMake(lastLexemInfoInTable, currBytePtr);

}

if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable) {

printf("\r\nError in the code generator! \"%s\" - unexpected token!\r\n", (\*lastLexemInfoInTable)->lexemStr);

exit(0);

}

}

currBytePtr = makeResetHWStack(lastLexemInfoInTable, currBytePtr, generatorMode);

currBytePtr = makeEndProgramCode(lastLexemInfoInTable, currBytePtr, generatorMode);

return currBytePtr;

}

//unsigned char outCode[GENERATED\_TEXT\_SIZE] = { '\0' };

void viewCode(unsigned char\* outCodePtr, unsigned long long int outCodePrintSize, unsigned char align) {

printf("\r\n; +0x0 +0x1 +0x2 +0x3 +0x4 +0x5 +0x6 +0x7 +0x8 +0x9 +0xA +0xB +0xC +0xD +0xE +0xF ");

printf("\r\n;0x00000000: ");

unsigned long long int outCodePrintIndex = outCodePrintSize - 1;

for (unsigned long long int index = 0; index <= outCodePrintIndex;) {

printf("0x%02X ", outCodePtr[index]);

if (!(++index % align)) {

unsigned long long int indexMinus16 = index - align;

do {

//printf("0x%02X ", outCodePtr[index]);

if (outCodePtr[indexMinus16] >= 32 && outCodePtr[indexMinus16] <= 126) {

printf("%c", outCodePtr[indexMinus16]);

}

else {

printf(" ");

//printf("%2c", 32);

}

} while (++indexMinus16 % align);

printf("\r\n;0x%08X: ", (unsigned int)index);

}

}

}  
  
goto\_label.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: goto\_lable.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string>

#include <map>

//#include <utility>

#include <stack>

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

std::map<std::string, std::pair<unsigned long long int, std::stack<unsigned long long int>>> labelInfoTable;

unsigned char\* makeLabelCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize, multitokenSize\_ = detectMultiToken(\*lastLexemInfoInTable + 1, MULTI\_TOKEN\_NULL\_STATEMENT);

multitokenSize = detectMultiToken(\*lastLexemInfoInTable + multitokenSize\_ + 1, MULTI\_TOKEN\_COLON);

if (multitokenSize) {

multitokenSize += multitokenSize\_;

}

if ((\*lastLexemInfoInTable)->tokenType != IDENTIFIER\_LEXEME\_TYPE){

return currBytePtr;

}

if (multitokenSize++) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;ident \"%s\"(as label) previous \"%s\"\r\n", (\*lastLexemInfoInTable)->lexemStr, tokenStruct[MULTI\_TOKEN\_COLON][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //ident \"%s\"(as label) previous \"%s\"\r\n", (\*lastLexemInfoInTable)->lexemStr, tokenStruct[MULTI\_TOKEN\_COLON][0]);

}

labelInfoTable[(\*lastLexemInfoInTable)->lexemStr].first = (unsigned long long int)currBytePtr;

while (!labelInfoTable[(\*lastLexemInfoInTable)->lexemStr].second.empty()) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)labelInfoTable[(\*lastLexemInfoInTable)->lexemStr].second.top() = (unsigned int)(currBytePtr - (unsigned char\*)labelInfoTable[(\*lastLexemInfoInTable)->lexemStr].second.top() - 4);

}

labelInfoTable[(\*lastLexemInfoInTable)->lexemStr].second.pop();

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@%016llX:\r\n", (unsigned long long int) & labelInfoTable[(\*lastLexemInfoInTable)->lexemStr].first);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_%016llX:\r\n", (unsigned long long int) & labelInfoTable[(\*lastLexemInfoInTable)->lexemStr].first);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeGotoLabelCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_GOTO);

if (multitokenSize++) {

if ((\*lastLexemInfoInTable + 1)->tokenType != IDENTIFIER\_LEXEME\_TYPE) {

return currBytePtr;

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" previous ident \"%s\"(as label)\r\n", tokenStruct[MULTI\_TOKEN\_GOTO][0], (\*lastLexemInfoInTable)[1].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" previous ident \"%s\"(as label)\r\n", tokenStruct[MULTI\_TOKEN\_GOTO][0], (\*lastLexemInfoInTable)[1].lexemStr);

}

if (labelInfoTable.find((\*lastLexemInfoInTable)[1].lexemStr) == labelInfoTable.end()) {

labelInfoTable[(\*lastLexemInfoInTable)[1].lexemStr].first = ~0;

}

if (labelInfoTable[(\*lastLexemInfoInTable)[1].lexemStr].first == ~0) {

labelInfoTable[(\*lastLexemInfoInTable)[1].lexemStr].second.push((unsigned long long int)(currBytePtr - 4));

}

else if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)((unsigned char\*)labelInfoTable[(\*lastLexemInfoInTable)[1].lexemStr].first - currBytePtr);

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@%016llX\r\n", (unsigned long long int) & labelInfoTable[(\*lastLexemInfoInTable)[1].lexemStr].first);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_%016llX;\r\n", (unsigned long long int) & labelInfoTable[(\*lastLexemInfoInTable)[1].lexemStr].first);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
greater.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: greater.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeIsGreaterCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_GREATER);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_cmp\_stackTopByECX\_eax[] = { 0x39, 0x01 };

const unsigned char code\_\_setg\_al[] = { 0x0F, 0x9F, 0xC0 };

const unsigned char code\_\_and\_eax\_1[] = { 0x83, 0xE0, 0x01 };

const unsigned char code\_\_mov\_stackTopByECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_stackTopByECX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_setg\_al, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_1, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_GREATER][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp dword ptr[ecx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " setg al\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and eax, 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr[ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_GREATER][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] = opStack[opStackIndex - 1] > opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

if\_then.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: if\_then.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

#include "string.h"

unsigned char\* makeIfCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_IF);

if (!multitokenSize

&& tokenStruct[MULTI\_TOKEN\_IF][1][0] == '('

&& !strncmp((\*lastLexemInfoInTable)->lexemStr, tokenStruct[MULTI\_TOKEN\_IF][0], MAX\_LEXEM\_SIZE)) {

multitokenSize = 1;

}

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_IF][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_IF][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeThenCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_THEN);

if (!multitokenSize && tokenStruct[MULTI\_TOKEN\_IF][1][0] == '(') {

multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_NULL\_STATEMENT);

}

if (multitokenSize

&& lexemInfoTransformationTempStackSize

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_IF][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_cmp\_eax\_0[] = { 0x83, 0xF8, 0x00 };

const unsigned char code\_\_jz\_offset[] = { 0x0F, 0x84, 0x00, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_eax\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jz\_offset, 6);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;after cond expresion (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_IF][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //after cond expresion (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_IF][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

/\*The lexem info table will be changed and will need to be rebuilt. TODO: change the implementation!\*/strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_THEN][0], MAX\_LEXEM\_SIZE);

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned long long int)(currBytePtr - 4);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, " cmp eax, 0\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jz LABEL@AFTER\_THEN\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " if (opTemp == 0) goto LABEL\_\_AFTER\_THEN\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makePostThenCode\_(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_1[] = { 0xB8, 0x01, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_1, 5);

\*(unsigned int\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned int)(currBytePtr - (unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue - 4);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, 1\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@AFTER\_THEN\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = 1;\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_AFTER\_THEN\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

return currBytePtr;

}

unsigned char\* makeSemicolonAfterThenCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) { // Or Ender!

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_SEMICOLON);

if (multitokenSize

&&

lexemInfoTransformationTempStackSize >= 2

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_IF][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_THEN][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\" (after \"then\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0], tokenStruct[MULTI\_TOKEN\_IF][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\" (after \"then\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0], tokenStruct[MULTI\_TOKEN\_IF][0]);

}

currBytePtr = makePostThenCode\_(lastLexemInfoInTable, currBytePtr, generatorMode);

lexemInfoTransformationTempStackSize -= 2;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
  
input.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: input.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeGetCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_INPUT);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_mov\_edx\_address[] = { 0xBA, 0x00, 0x00, 0x00, 0x00 };

const unsigned char code\_\_add\_edx\_esi[] = { 0x03, 0xD6 };

const unsigned char code\_\_push\_ecx[] = { 0x51 };

//const unsigned char code\_\_push\_ebx[] = { 0x53 };

const unsigned char code\_\_push\_esi[] = { 0x56 };

const unsigned char code\_\_push\_edi[] = { 0x57 };

const unsigned char code\_\_call\_edx[] = { 0xFF, 0xD2 };

const unsigned char code\_\_pop\_edi[] = { 0x5F };

const unsigned char code\_\_pop\_esi[] = { 0x5E };

//const unsigned char code\_\_pop\_ebx[] = { 0x5B };

const unsigned char code\_\_pop\_ecx[] = { 0x59 };

const unsigned char code\_\_mov\_ebx\_valueByAdrressInECX[] = { 0x8B, 0x19 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_add\_ebx\_edi[] = { 0x33, 0xDF };

const unsigned char code\_\_mov\_stackTopByEBX\_eax[] = { 0x89, 0x03 };

const unsigned char code\_\_mov\_ecx\_edi[] = { 0x8B, 0xCF };

const unsigned char code\_\_add\_ecx\_512[] = { 0x81, 0xC1, 0x00, 0x02, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_edx\_address, 5);

\*(unsigned int\*)&(currBytePtr[-4]) = (unsigned int)getProcOffset;

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_edx\_esi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_ecx, 1);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_ebx, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_esi, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_edi, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_call\_edx, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_edi, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_esi, 1);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_ebx, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_ecx, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ebx\_valueByAdrressInECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ebx\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByEBX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ecx\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ecx\_512, 6);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_INPUT][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " mov edx, 0%08Xh\r\n", (unsigned int)getProcOffset);

currBytePtr += sprintf((char\*)currBytePtr, " add edx, esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call edx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " pop edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " pop esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;pop ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " pop ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ebx, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ebx, edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ebx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ecx, edi ; reset second stack\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ecx, 512 ; reset second stack\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_INPUT][0]);

currBytePtr += sprintf((char\*)currBytePtr, " (void)scanf\_s(\"%%d\", &opTemp);\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " data[opStack[opStackIndex]] = opTemp, opStackIndex = 0;\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
less.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: less.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeIsLessCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_LESS);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_cmp\_stackTopByECX\_eax[] = { 0x39, 0x01 };

const unsigned char code\_\_setl\_al[] = { 0x0F, 0x9C, 0xC0 };

const unsigned char code\_\_and\_eax\_1[] = { 0x83, 0xE0, 0x01 };

const unsigned char code\_\_mov\_stackTopByECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_stackTopByECX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_setl\_al, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_1, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_LESS][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp dword ptr[ecx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " setl al\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and eax, 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr[ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_LESS][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] = opStack[opStackIndex - 1] < opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
lexica.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: lexica.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/config.h"

#include "../../../src/include/lexica/lexica.h"

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include <fstream>

#include <iostream>

//#include <algorithm>

#include <iterator>

#include <regex>

//struct LexemInfo {

// char lexemStr[MAX\_LEXEM\_SIZE];

// unsigned int lexemId;

// unsigned int tokenType;

// unsigned int ifvalue;

// unsigned int row;

// unsigned int col;

// // TODO: ...

//};

#define MAX\_ACCESSORY\_STACK\_SIZE\_123 128

char tempStrFor\_123[MAX\_TEXT\_SIZE/\*?TODO:... MAX\_ACCESSORY\_STACK\_SIZE\_123 \* 64\*/] = {'\0'};

unsigned long long int tempStrForCurrIndex = 0;

struct LexemInfo lexemesInfoTable[MAX\_WORD\_COUNT];// = { { "", 0, 0, 0 } };

struct LexemInfo\* lastLexemInfoInTable = lexemesInfoTable; // first for begin

char identifierIdsTable[MAX\_WORD\_COUNT][MAX\_LEXEM\_SIZE] = { "" };

LexemInfo::LexemInfo() {

lexemStr[0] = '\0';

lexemId = 0;

tokenType = 0;

ifvalue = 0;

row = ~0;

col = ~0;

}

LexemInfo::LexemInfo(const char \* lexemStr, unsigned long long int lexemId, unsigned long long int tokenType, unsigned long long int ifvalue, unsigned long long int row, unsigned long long int col) {

strncpy(this->lexemStr, lexemStr, MAX\_LEXEM\_SIZE);

this->lexemId = lexemId;

this->tokenType = tokenType;

this->ifvalue = ifvalue;

this->row = row;

this->col = col;

}

LexemInfo::LexemInfo(const NonContainedLexemInfo& nonContainedLexemInfo){

strncpy(lexemStr, nonContainedLexemInfo.lexemStr, MAX\_LEXEM\_SIZE);

lexemId = nonContainedLexemInfo.lexemId;

tokenType = nonContainedLexemInfo.tokenType;

ifvalue = nonContainedLexemInfo.ifvalue;

row = nonContainedLexemInfo.row;

col = nonContainedLexemInfo.col;

}

NonContainedLexemInfo::NonContainedLexemInfo() {

(lexemStr = tempStrFor\_123 + tempStrForCurrIndex)[0] = '\0';

tempStrForCurrIndex += 32;// MAX\_LEXEM\_SIZE;

lexemId = 0;

tokenType = 0;

ifvalue = 0;

row = ~0;

col = ~0;

}

NonContainedLexemInfo::NonContainedLexemInfo(const LexemInfo& lexemInfo) {

//strncpy(lexemStr, lexemInfo.lexemStr, MAX\_LEXEM\_SIZE); //

lexemStr = (char\*)lexemInfo.lexemStr;

lexemId = lexemInfo.lexemId;

tokenType = lexemInfo.tokenType;

ifvalue = lexemInfo.ifvalue;

row = lexemInfo.row;

col = lexemInfo.col;

}

void printLexemes(struct LexemInfo\* lexemInfoTable, char printBadLexeme) {

if (printBadLexeme) {

printf("Bad lexeme:\r\n");

}

else {

printf("Lexemes table:\r\n");

}

printf("-------------------------------------------------------------------\r\n");

//printf("index\t\tlexeme\t\tid\ttype\tifvalue\trow\tcol\r\n");

printf("index lexeme id type ifvalue row col\r\n");

printf("-------------------------------------------------------------------\r\n");

for (unsigned long long int index = 0; (!index || !printBadLexeme) && lexemInfoTable[index].lexemStr[0] != '\0'; ++index) {

printf("%5llu%17s%12llu%10llu%11llu%4lld%8lld\r\n", index, lexemInfoTable[index].lexemStr, lexemInfoTable[index].lexemId, lexemInfoTable[index].tokenType, lexemInfoTable[index].ifvalue, lexemInfoTable[index].row, lexemInfoTable[index].col);

}

printf("-------------------------------------------------------------------\r\n\r\n");

return;

}

void printLexemesToFile(struct LexemInfo\* lexemInfoTable, char printBadLexeme, const char\* filename) {

FILE\* file = fopen(filename, "wb");

if (!file) {

perror("Failed to open file");

return;

}

if (printBadLexeme) {

fprintf(file, "Bad lexeme:\r\n");

}

else {

fprintf(file, "Lexemes table:\r\n");

}

fprintf(file, "-------------------------------------------------------------------\r\n");

//fprintf(file, "index\t\tlexeme\t\tid\ttype\tifvalue\trow\tcol\r\n");

fprintf(file, "index lexeme id type ifvalue row col\r\n");

fprintf(file, "-------------------------------------------------------------------\r\n");

for (unsigned long long int index = 0; (!index || !printBadLexeme) && lexemInfoTable[index].lexemStr[0] != '\0'; ++index) {

fprintf(file, "%5llu%17s%12llu%10llu%11llu%4lld%8lld\r\n",

index,

lexemInfoTable[index].lexemStr,

lexemInfoTable[index].lexemId,

lexemInfoTable[index].tokenType,

lexemInfoTable[index].ifvalue,

lexemInfoTable[index].row,

lexemInfoTable[index].col);

}

fprintf(file, "-------------------------------------------------------------------\r\n\r\n");

fclose(file);

}

// get identifier id

unsigned int getIdentifierId(char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\* str) {

unsigned int index = 0;

for (; identifierIdsTable[index][0] != '\0'; ++index) {

if (!strncmp(identifierIdsTable[index], str, MAX\_LEXEM\_SIZE)) {

return index;

}

}

strncpy(identifierIdsTable[index], str, MAX\_LEXEM\_SIZE);

identifierIdsTable[index + 1][0] = '\0'; // not necessarily for zero-init identifierIdsTable

return index;

}

// try to get identifier

unsigned int tryToGetIdentifier(struct LexemInfo\* lexemInfoInTable, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE]) {

char identifiers\_re[] = IDENTIFIERS\_RE;

//char identifiers\_re[] = "\_[A-Z][A-Z][A-Z][A-Z][A-Z][A-Z][A-Z]";

if (std::regex\_match(std::string(lexemInfoInTable->lexemStr), std::regex(identifiers\_re))) {

lexemInfoInTable->lexemId = getIdentifierId(identifierIdsTable, lexemInfoInTable->lexemStr);

lexemInfoInTable->tokenType = IDENTIFIER\_LEXEME\_TYPE;

return SUCCESS\_STATE;

}

return ~SUCCESS\_STATE;

}

// try to get value

unsigned int tryToGetUnsignedValue(struct LexemInfo\* lexemInfoInTable) {

char unsignedvalues\_re[] = UNSIGNEDVALUES\_RE;

//char unsignedvalues\_re[] = "0|[1-9][0-9]\*";

if (std::regex\_match(std::string(lexemInfoInTable->lexemStr), std::regex(unsignedvalues\_re))) {

lexemInfoInTable->ifvalue = atoi(lastLexemInfoInTable->lexemStr);

lexemInfoInTable->lexemId = MAX\_VARIABLES\_COUNT + MAX\_KEYWORD\_COUNT;

lexemInfoInTable->tokenType = VALUE\_LEXEME\_TYPE;

return SUCCESS\_STATE;

}

return ~SUCCESS\_STATE;

}

int commentRemover(char\* text, const char\* openStrSpc, const char\* closeStrSpc) {

bool eofAlternativeCloseStrSpcType = false;

bool explicitCloseStrSpc = true;

if (!strcmp(closeStrSpc, "\n")) {

eofAlternativeCloseStrSpcType = true;

explicitCloseStrSpc = false;

}

unsigned int commentSpace = 0;

unsigned int textLength = strlen(text); // strnlen(text, MAX\_TEXT\_SIZE);

unsigned int openStrSpcLength = strlen(openStrSpc); // strnlen(openStrSpc, MAX\_TEXT\_SIZE);

unsigned int closeStrSpcLength = strlen(closeStrSpc); // strnlen(closeStrSpc, MAX\_TEXT\_SIZE);

if (!closeStrSpcLength) {

return -1; // no set closeStrSpc

}

unsigned char oneLevelComment = 0;

if (!strncmp(openStrSpc, closeStrSpc, MAX\_LEXEM\_SIZE)) {

oneLevelComment = 1;

}

for (unsigned int index = 0; index < textLength; ++index) {

if (!strncmp(text + index, closeStrSpc, closeStrSpcLength) && (explicitCloseStrSpc || commentSpace)) {

if (commentSpace == 1 && explicitCloseStrSpc) {

for (unsigned int index2 = 0; index2 < closeStrSpcLength; ++index2) {

text[index + index2] = ' ';

}

}

else if (commentSpace == 1 && !explicitCloseStrSpc) {

index += closeStrSpcLength - 1;

}

oneLevelComment ? commentSpace = !commentSpace : commentSpace = 0;

}

else if (!strncmp(text + index, openStrSpc, openStrSpcLength)) {

oneLevelComment ? commentSpace = !commentSpace : commentSpace = 1;

}

if (commentSpace && text[index] != ' ' && text[index] != '\t' && text[index] != '\r' && text[index] != '\n') {

text[index] = ' ';

}

}

if (commentSpace && !eofAlternativeCloseStrSpcType) {

return -1;

}

return 0;

}

void prepareKeyWordIdGetter(char\* keywords\_, char\* keywords\_re) {

if (keywords\_ == NULL || keywords\_re == NULL) {

return;

}

for (char\* keywords\_re\_ = keywords\_re, \*keywords\_\_ = keywords\_; (\*keywords\_re\_ != '\0') ? 1 : (\*keywords\_\_ = '\0', 0); (\*keywords\_re\_ != '\\' || (keywords\_re\_[1] != '+' && keywords\_re\_[1] != '\*' && keywords\_re\_[1] != '|')) ? \*keywords\_\_++ = \*keywords\_re\_ : 0, ++keywords\_re\_);

}

unsigned int getKeyWordId(char\* keywords\_, char\* lexemStr, unsigned int baseId) {

if (keywords\_ == NULL || lexemStr == NULL) {

return ~0;

}

char\* lexemInKeywords\_ = keywords\_;

size\_t lexemStrLen = strlen(lexemStr);

if (!lexemStrLen) {

return ~0;

}

for (; lexemInKeywords\_ = strstr(lexemInKeywords\_, lexemStr), lexemInKeywords\_ != NULL && lexemInKeywords\_[lexemStrLen] != '|' && lexemInKeywords\_[lexemStrLen] != '\0'; ++lexemInKeywords\_);

return lexemInKeywords\_ - keywords\_ + baseId;

}

// try to get KeyWord

char tryToGetKeyWord(struct LexemInfo\* lexemInfoInTable) {

char keywords\_re[] = KEYWORDS\_RE;

//char keywords\_re[] = ";|<<|>>|\\+|-|\\\*|,|==|!=|:|\\(|\\)|NAME|DATA|BODY|END|EXIT|CONTINUE|GET|PUT|IF|ELSE|FOR|TO|DOWNTO|DO|WHILE|REPEAT|UNTIL|GOTO|DIV|MOD|<=|>=|NOT|AND|OR|INTEGER16";

//char keywords\_re[] = ";|<<|\\+\\+|--|\\\*\\\*|==|\\(|\\)|!=|:|name|data|body|end|get|put|for|to|downto|do|while|continue|exit|repeat|until|if|goto|div|mod|le|ge|not|and|or|long|int";

char keywords\_[sizeof(keywords\_re)] = { '\0' };

prepareKeyWordIdGetter(keywords\_, keywords\_re);

if (std::regex\_match(std::string(lexemInfoInTable->lexemStr), std::regex(keywords\_re))) {

lexemInfoInTable->lexemId = getKeyWordId(keywords\_, lexemInfoInTable->lexemStr, MAX\_VARIABLES\_COUNT);

lexemInfoInTable->tokenType = KEYWORD\_LEXEME\_TYPE;

return SUCCESS\_STATE;

}

return ~SUCCESS\_STATE;

}

void setPositions(const char\* text, struct LexemInfo\* lexemInfoTable) {

unsigned long long int line\_number = 1;

const char\* pos = text, \* line\_start = text;

if (lexemInfoTable) while (\*pos != '\0' && lexemInfoTable->lexemStr[0] != '\0') {

const char\* line\_end = strchr(pos, '\n');

if (!line\_end) {

line\_end = text + strlen(text);

}

char line\_[4096], \* line = line\_; //!! TODO: ...

strncpy(line, pos, line\_end - pos);

line[line\_end - pos] = '\0';

for (char\* found\_pos; lexemInfoTable->lexemStr[0] != '\0' && (found\_pos = strstr(line, lexemInfoTable->lexemStr)); line += strlen(lexemInfoTable->lexemStr), ++lexemInfoTable) {

lexemInfoTable->row = line\_number;

lexemInfoTable->col = found\_pos - line\_ + 1ull;

}

line\_number++;

pos = line\_end;

if (\*pos == '\n') {

pos++;

}

}

}

struct LexemInfo lexicalAnalyze(struct LexemInfo\* lexemInfoInPtr, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE]) {

struct LexemInfo ifBadLexemeInfo; // = { 0 };

if (tryToGetKeyWord(lexemInfoInPtr) == SUCCESS\_STATE);

else if (tryToGetIdentifier(lexemInfoInPtr, identifierIdsTable) == SUCCESS\_STATE);

else if (tryToGetUnsignedValue(lexemInfoInPtr) == SUCCESS\_STATE);

else {

ifBadLexemeInfo.tokenType = UNEXPEXTED\_LEXEME\_TYPE;

}

return ifBadLexemeInfo;

}

struct LexemInfo tokenize(char\* text, struct LexemInfo\*\* lastLexemInfoInTable, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], struct LexemInfo(\*lexicalAnalyzeFunctionPtr)(struct LexemInfo\*, char(\*)[MAX\_LEXEM\_SIZE])) {

char tokens\_re[] = TOKENS\_RE;

//char tokens\_re[] = ";|<<|>>|\\+|-|\\\*|,|==|!=|:|\\(|\\)|<=|>=|[\_0-9A-Za-z]+|[^ \t\r\f\v\n]";

//char tokens\_re[] = "<<|\\+\\+|--|\\\*\\\*|==|\\(|\\)|!=|[\_0-9A-Za-z]+|[^ \t\r\f\v\n]";

std::regex tokens\_re\_(tokens\_re);

struct LexemInfo ifBadLexemeInfo; // = { 0 };

std::string stringText(text);

for (std::sregex\_token\_iterator end, tokenIterator(stringText.begin(), stringText.end(), tokens\_re\_); tokenIterator != end; ++tokenIterator, ++ \* lastLexemInfoInTable) {

std::string str = \*tokenIterator;

strncpy((\*lastLexemInfoInTable)->lexemStr, str.c\_str(), MAX\_LEXEM\_SIZE);

if ((ifBadLexemeInfo = (\*lexicalAnalyzeFunctionPtr)(\*lastLexemInfoInTable, identifierIdsTable)).tokenType == UNEXPEXTED\_LEXEME\_TYPE) {

break;

}

}

setPositions(text, lexemesInfoTable);

if (ifBadLexemeInfo.tokenType == UNEXPEXTED\_LEXEME\_TYPE) {

strncpy(ifBadLexemeInfo.lexemStr, (\*lastLexemInfoInTable)->lexemStr, MAX\_LEXEM\_SIZE);

ifBadLexemeInfo.row = (\*lastLexemInfoInTable)->row;

ifBadLexemeInfo.col = (\*lastLexemInfoInTable)->col;

}

return ifBadLexemeInfo;

}  
  
lrbind.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: lrbind.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeLeftToRightBindCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_LRBIND);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_ebx\_stackTopByECX[] = { 0x8B, 0x19 };

const unsigned char code\_\_mov\_eax\_stackTopByECXMinus4[] = { 0x8B, 0x41, 0xFC };

const unsigned char code\_\_sub\_ecx\_8[] = { 0x83, 0xE9, 0x08 };

const unsigned char code\_\_add\_ebx\_edi[] = { 0x03, 0xDF };

const unsigned char code\_\_mov\_addrFromEBX\_eax[] = { 0x89, 0x03 };

const unsigned char code\_\_mov\_ecx\_edi[] = { 0x8B, 0xCF };

const unsigned char code\_\_add\_ecx\_512[] = { 0x81, 0xC1, 0x00, 0x02, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ebx\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECXMinus4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_8, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ebx\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_addrFromEBX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ecx\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ecx\_512, 6);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_LRBIND][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov ebx, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx - 4]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 8\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ebx, edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ebx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ecx, edi ; reset second stack\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ecx, 512 ; reset second stack\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_LRBIND][0]);

currBytePtr += sprintf((char\*)currBytePtr, " lastBindDataIndex = opStack[opStackIndex];\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " data[lastBindDataIndex] = opTemp = opStack[opStackIndex - 1], opStackIndex = 0;\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
machinecodegen\_addon.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: machinecodegen\_addon.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

//#define DEBUG\_RECONSTRUCT

unsigned char buffer[256 \* 1024];

extern unsigned long long int a123\_array\_part\_count;

extern unsigned char\* a123[39];

extern unsigned long long int a123\_array\_part\_size[39];

extern unsigned long long int a123\_zero\_part\_count;

extern unsigned long long int a123\_zeros[39];

extern unsigned long long int o123\_array\_part\_count;

extern unsigned char\* o123[5];

extern unsigned long long int o123\_array\_part\_size[5];

extern unsigned long long int o123\_zero\_part\_count;

extern unsigned long long int o123\_zeros[4];

unsigned long long int buildTemplateForCodeObject(unsigned char\* byteImage) {

if (!byteImage) {

exit(EXIT\_FAILURE);

}

unsigned long long int byteIndex = 0;

unsigned long long int totalByteCount = 0;

unsigned long long int array\_index = 0;

unsigned long long int zero\_index = 0;

for (unsigned long long int i = 0; i < o123\_array\_part\_count + o123\_zero\_part\_count; ++i) {

if (i % 2 == 0) {

unsigned char\* current\_array = NULL;

current\_array = o123[array\_index++];

if (current\_array) {

//fwrite(current\_array, 1, o123\_array\_part\_size[array\_index - 1]/\*sizeof(current\_array)\*/, outfile);

for (unsigned long long int localByteIndex = 0; localByteIndex < o123\_array\_part\_size[array\_index - 1]; ++localByteIndex) {

byteImage[byteIndex++] = current\_array[localByteIndex];

}

#ifdef DEBUG\_RECONSTRUCT

printf(": sizeof(current\_array) = %llu\r\n", o123\_array\_part\_size[array\_index - 1]/\*sizeof(current\_array)\*/);

#endif

totalByteCount += o123\_array\_part\_size[array\_index - 1];

}

}

else {

unsigned long long int zero\_count = 0;

zero\_count = o123\_zeros[zero\_index++];

if (zero\_count > 0) {

unsigned char\* zeros = (unsigned char\*)calloc(zero\_count, sizeof(unsigned char));

if (!zeros) {

perror("Memory allocation failed for zero block");

//fclose(outfile);

exit(EXIT\_FAILURE);

}

//fwrite(zeros, 1, zero\_count, outfile);

for (unsigned long long int localByteIndex = 0; localByteIndex < zero\_count; ++localByteIndex) {

byteImage[byteIndex++] = 0;

}

#ifdef DEBUG\_RECONSTRUCT

printf("zero\_count = %llu\r\n", zero\_count);

#endif

totalByteCount += zero\_count;

free(zeros);

}

}

}

//fclose(outfile);

#ifdef DEBUG\_RECONSTRUCT

printf("Image reconstructed.\n"/\*, output\_file\*/);

#endif

return byteIndex;

}

unsigned long long int buildTemplateForCodeImage(unsigned char\* byteImage) {

if (!byteImage) {

exit(EXIT\_FAILURE);

}

unsigned long long int byteIndex = 0;

unsigned long long int totalByteCount = 0;

unsigned long long int array\_index = 0;

unsigned long long int zero\_index = 0;

for (unsigned long long int i = 0; i < a123\_array\_part\_count + a123\_zero\_part\_count; ++i) {

if (i % 2 == 0) {

unsigned char\* current\_array = NULL;

current\_array = a123[array\_index++];

if (current\_array) {

//fwrite(current\_array, 1, a123\_array\_part\_size[array\_index - 1]/\*sizeof(current\_array)\*/, outfile);

for (unsigned long long int localByteIndex = 0; localByteIndex < a123\_array\_part\_size[array\_index - 1]; ++localByteIndex) {

byteImage[byteIndex++] = current\_array[localByteIndex];

}

#ifdef DEBUG\_RECONSTRUCT

printf(": sizeof(current\_array) = %llu\r\n", a123\_array\_part\_size[array\_index - 1]/\*sizeof(current\_array)\*/);

#endif

totalByteCount += a123\_array\_part\_size[array\_index - 1];

}

}

else {

unsigned long long int zero\_count = 0;

zero\_count = a123\_zeros[zero\_index++];

if (zero\_count > 0) {

unsigned char\* zeros = (unsigned char\*)calloc(zero\_count, sizeof(unsigned char));

if (!zeros) {

perror("Memory allocation failed for zero block");

//fclose(outfile);

exit(EXIT\_FAILURE);

}

//fwrite(zeros, 1, zero\_count, outfile);

for (unsigned long long int localByteIndex = 0; localByteIndex < zero\_count; ++localByteIndex) {

byteImage[byteIndex++] = 0;

}

#ifdef DEBUG\_RECONSTRUCT

printf("zero\_count = %llu\r\n", zero\_count);

#endif

totalByteCount += zero\_count;

free(zeros);

}

}

}

//fclose(outfile);

#ifdef DEBUG\_RECONSTRUCT

printf("Image reconstructed.\n"/\*, output\_file\*/);

#endif

return byteIndex;

}

void writeBytesToFile(const char\* output\_file, unsigned char\* byteImage, unsigned long long int imageSize) {

if (!output\_file || !byteImage) {

perror("Error in write image to file");

exit(EXIT\_FAILURE);

}

FILE\* outfile = fopen(output\_file, "wb");

if (!outfile) {

perror("Error opening output file");

exit(EXIT\_FAILURE);

}

if (imageSize) {

fwrite(byteImage, 1, imageSize, outfile);

}

fclose(outfile);

printf("File \"%s\" saved.\n", output\_file);

}  
  
  
machinecodegen\_pattern.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: machinecodegen\_pattern.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char a123\_0[] = {

0x4D,0x5A,0x90,0x00,0x03,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,

0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,

};

#define a123\_0\_SIZE 25

#define a123\_ZEROS\_0 35

unsigned char a123\_1[] = {

0xD0,0x00,0x00,0x00,0x0E,0x1F,0xBA,0x0E,0x00,0xB4,0x09,0xCD,0x21,0xB8,0x01,0x4C,

0xCD,0x21,0x54,0x68,0x69,0x73,0x20,0x70,0x72,0x6F,0x67,0x72,0x61,0x6D,0x20,0x63,

0x61,0x6E,0x6E,0x6F,0x74,0x20,0x62,0x65,0x20,0x72,0x75,0x6E,0x20,0x69,0x6E,0x20,

0x44,0x4F,0x53,0x20,0x6D,0x6F,0x64,0x65,0x2E,0x0D,0x0D,0x0A,0x24,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0xBD,0x32,0xF9,0xFD,0xF9,0x53,0x97,0xAE,0xF9,0x53,0x97,0xAE,

0xF9,0x53,0x97,0xAE,0xED,0x38,0x96,0xAF,0xFC,0x53,0x97,0xAE,0xF9,0x53,0x96,0xAE,

0xFD,0x53,0x97,0xAE,0x1D,0x27,0x94,0xAF,0xF8,0x53,0x97,0xAE,0x1D,0x27,0x68,0xAE,

0xF8,0x53,0x97,0xAE,0x1D,0x27,0x95,0xAF,0xF8,0x53,0x97,0xAE,0x52,0x69,0x63,0x68,

0xF9,0x53,0x97,0xAE,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x50,0x45,0x00,0x00,0x4C,0x01,0x05,0x00,0xCA,0x22,0x65,0x67,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xE0,0x00,0x02,0x01,0x0B,0x01,0x0E,0x1C,

0x00,0x22,0x00,0x00,0x00,0x2A,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x00,0x00,

0x00,0x10,0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x00,0x40,0x00,0x00,0x10,0x00,0x00,

0x00,0x02,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0xA0,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x00,0x00,0x00,

0x03,0x00,0x40,0x81,0x00,0x00,0x10,0x00,0x00,0x10,0x00,0x00,0x00,0x00,0x10,0x00,

0x00,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0xCC,0x41,0x00,0x00,0x3C,0x00,0x00,0x00,0x00,0x80,0x00,0x00,

0xE0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x90,0x00,0x00,0x30,0x00,0x00,0x00,0x18,0x40,0x00,0x00,

0x70,

};

#define a123\_1\_SIZE 321

#define a123\_ZEROS\_1 44

unsigned char a123\_2[] = {

0x40,0x00,0x00,0x18,

};

#define a123\_2\_SIZE 4

#define a123\_ZEROS\_2 27

unsigned char a123\_3[] = {

0x2E,0x74,0x65,0x78,0x74,0x00,0x00,0x00,0xCA,0x20,0x00,0x00,0x00,0x10,0x00,0x00,

0x00,0x22,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x20,0x00,0x00,0x60,0x2E,0x72,0x64,0x61,0x74,0x61,0x00,0x00,

0x76,0x02,0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x26,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,0x00,0x00,0x40,

0x2E,0x64,0x61,0x74,0x61,0x00,0x00,0x00,0x91,0x21,0x00,0x00,0x00,0x50,0x00,0x00,

0x00,0x22,0x00,0x00,0x00,0x2A,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x40,0x00,0x00,0xC0,0x2E,0x72,0x73,0x72,0x63,0x00,0x00,0x00,

0xE0,0x01,0x00,0x00,0x00,0x80,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x4C,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,0x00,0x00,0x40,

0x2E,0x72,0x65,0x6C,0x6F,0x63,0x00,0x00,0x30,0x00,0x00,0x00,0x00,0x90,0x00,0x00,

0x00,0x02,0x00,0x00,0x00,0x4E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x40,0x00,0x00,0x42,

};

#define a123\_3\_SIZE 200

#define a123\_ZEROS\_3 368

unsigned char a123\_4[] = {

0xE8,0x00,0x00,0x00,0x00,0x5E,0x83,0xEE,0x05,0x8B,0xFE,0x81,0xC7,0x00,0x40,0x00,

0x00,0x8B,0xCF,0x81,0xC1,0x00,0x02,0x00,0x00,0xEB,0x7C,0x50,0x68,0x00,0x71,0x40,

0x00,0x68,0x00,0x70,0x40,0x00,0xE8,0x99,0x20,0x00,0x00,0x83,0xC4,0x0C,0x6A,0x00,

0x6A,0x00,0x50,0x68,0x00,0x70,0x40,0x00,0xFF,0x35,0x09,0x71,0x40,0x00,0xE8,0x7B,

0x20,0x00,0x00,0xC3,0x55,0x8B,0xEC,0x6A,0x00,0x68,0x8D,0x71,0x40,0x00,0x6A,0x0F,

0x68,0x0E,0x71,0x40,0x00,0xFF,0x35,0x05,0x71,0x40,0x00,0xE8,0x58,0x20,0x00,0x00,

0x8D,0x35,0x0D,0x71,0x40,0x00,0x03,0x35,0x8D,0x71,0x40,0x00,0x83,0xEE,0x02,0xE8,

0x04,0x00,0x00,0x00,0x8B,0xE5,0x5D,0xC3,0x33,0xC0,0xBB,0x01,0x00,0x00,0x00,0x33,

0xC9,0x0F,0xB6,0x0E,0x85,0xC9,0x74,0x0E,0x83,0xE9,0x30,0x0F,0xAF,0xCB,0x03,0xC1,

0x6B,0xDB,0x0A,0x4E,0xEB,0xEB,0xC3,0x6A,0xF6,0xE8,0x14,0x20,0x00,0x00,0xA3,0x05,

0x71,0x40,0x00,0x6A,0xF5,0xE8,0x08,0x20,0x00,0x00,0xA3,0x09,0x71,0x40,0x00,0x33,

0xC0,0xC3,

};

#define a123\_4\_SIZE 178

#define a123\_ZEROS\_4 8192

unsigned char a123\_5[] = {

0xFF,0x25,0x08,0x40,0x40,0x00,0xFF,0x25,0x00,0x40,0x40,0x00,0xFF,0x25,0x04,0x40,

0x40,0x00,0xFF,0x25,0x10,0x40,0x40,

};

#define a123\_5\_SIZE 23

#define a123\_ZEROS\_5 311

unsigned char a123\_6[] = {

0x30,0x42,0x00,0x00,0x40,0x42,0x00,0x00,0x20,0x42,0x00,0x00,0x00,0x00,0x00,0x00,

0x5E,0x42,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xCA,0x22,0x65,0x67,

0x00,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x54,0x00,0x00,0x00,0x88,0x40,0x00,0x00,

0x88,0x26,0x00,0x00,0x00,0x00,0x00,0x00,0xCA,0x22,0x65,0x67,0x00,0x00,0x00,0x00,

0x0C,0x00,0x00,0x00,0x14,0x00,0x00,0x00,0xDC,0x40,0x00,0x00,0xDC,0x26,0x00,0x00,

0x00,0x00,0x00,0x00,0xCA,0x22,0x65,0x67,0x00,0x00,0x00,0x00,0x0D,0x00,0x00,0x00,

0xDC,0x00,0x00,0x00,0xF0,0x40,0x00,0x00,0xF0,0x26,0x00,0x00,0x00,0x00,0x00,0x00,

0xCA,0x22,0x65,0x67,0x00,0x00,0x00,0x00,0x0E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x52,0x53,0x44,0x53,0x05,0x39,0xA1,0x32,

0x25,0x9B,0x91,0x44,0x8E,0x6F,0x3B,0x20,0xBC,0x79,0x50,0x25,0x22,0x00,0x00,0x00,

0x43,0x3A,0x5C,0x55,0x73,0x65,0x72,0x73,0x5C,0x4E,0x61,0x7A,0x61,0x72,0x5C,0x73,

0x6F,0x75,0x72,0x63,0x65,0x5C,0x72,0x65,0x70,0x6F,0x73,0x5C,0x50,0x72,0x6F,0x6A,

0x65,0x63,0x74,0x36,0x30,0x5C,0x52,0x65,0x6C,0x65,0x61,0x73,0x65,0x5C,0x50,0x72,

0x6F,0x6A,0x65,0x63,0x74,0x36,0x30,0x2E,0x70,0x64,0x62,

};

#define a123\_6\_SIZE 219

#define a123\_ZEROS\_6 21

unsigned char a123\_7[] = {

0x47,0x43,0x54,0x4C,0x00,0x10,0x00,0x00,0xCA,0x20,0x00,0x00,0x2E,0x74,0x65,0x78,

0x74,0x24,0x6D,0x6E,0x00,0x00,0x00,0x00,0x00,0x40,0x00,0x00,0x18,0x00,0x00,0x00,

0x2E,0x69,0x64,0x61,0x74,0x61,0x24,0x35,0x00,0x00,0x00,0x00,0x18,0x40,0x00,0x00,

0x70,0x00,0x00,0x00,0x2E,0x72,0x64,0x61,0x74,0x61,0x00,0x00,0x88,0x40,0x00,0x00,

0x44,0x01,0x00,0x00,0x2E,0x72,0x64,0x61,0x74,0x61,0x24,0x7A,0x7A,0x7A,0x64,0x62,

0x67,0x00,0x00,0x00,0xCC,0x41,0x00,0x00,0x28,0x00,0x00,0x00,0x2E,0x69,0x64,0x61,

0x74,0x61,0x24,0x32,0x00,0x00,0x00,0x00,0xF4,0x41,0x00,0x00,0x14,0x00,0x00,0x00,

0x2E,0x69,0x64,0x61,0x74,0x61,0x24,0x33,0x00,0x00,0x00,0x00,0x08,0x42,0x00,0x00,

0x18,0x00,0x00,0x00,0x2E,0x69,0x64,0x61,0x74,0x61,0x24,0x34,0x00,0x00,0x00,0x00,

0x20,0x42,0x00,0x00,0x56,0x00,0x00,0x00,0x2E,0x69,0x64,0x61,0x74,0x61,0x24,0x36,

0x00,0x00,0x00,0x00,0x00,0x50,0x00,0x00,0x91,0x21,0x00,0x00,0x2E,0x64,0x61,0x74,

0x61,0x00,0x00,0x00,0x00,0x80,0x00,0x00,0x60,0x00,0x00,0x00,0x2E,0x72,0x73,0x72,

0x63,0x24,0x30,0x31,0x00,0x00,0x00,0x00,0x60,0x80,0x00,0x00,0x80,0x01,0x00,0x00,

0x2E,0x72,0x73,0x72,0x63,0x24,0x30,0x32,0x00,0x00,0x00,0x00,0x08,0x42,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x50,0x42,0x00,0x00,0x00,0x40,0x00,0x00,

0x18,0x42,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x6A,0x42,0x00,0x00,

0x10,0x40,

};

#define a123\_7\_SIZE 258

#define a123\_ZEROS\_7 22

unsigned char a123\_8[] = {

0x30,0x42,0x00,0x00,0x40,0x42,0x00,0x00,0x20,0x42,0x00,0x00,0x00,0x00,0x00,0x00,

0x5E,0x42,0x00,0x00,0x00,0x00,0x00,0x00,0xD5,0x02,0x47,0x65,0x74,0x53,0x74,0x64,

0x48,0x61,0x6E,0x64,0x6C,0x65,0x00,0x00,0x68,0x04,0x52,0x65,0x61,0x64,0x43,0x6F,

0x6E,0x73,0x6F,0x6C,0x65,0x41,0x00,0x00,0x0B,0x06,0x57,0x72,0x69,0x74,0x65,0x43,

0x6F,0x6E,0x73,0x6F,0x6C,0x65,0x41,0x00,0x4B,0x45,0x52,0x4E,0x45,0x4C,0x33,0x32,

0x2E,0x64,0x6C,0x6C,0x00,0x00,0xE1,0x03,0x77,0x73,0x70,0x72,0x69,0x6E,0x74,0x66,

0x41,0x00,0x55,0x53,0x45,0x52,0x33,0x32,0x2E,0x64,0x6C,0x6C,

};

#define a123\_8\_SIZE 108

#define a123\_ZEROS\_8 8844

unsigned char a123\_9[] = {

0x25,0x64,0x0A,0x0D,

};

#define a123\_9\_SIZE 4

#define a123\_ZEROS\_9 266

unsigned char a123\_10[] = {

0x01,0x00,0x18,0x00,0x00,0x00,0x18,0x00,0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x01,0x00,0x00,0x00,0x30,0x00,

0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x01,0x00,0x09,0x04,0x00,0x00,0x48,0x00,0x00,0x00,0x60,0x80,0x00,0x00,0x7D,0x01,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x3C,0x3F,0x78,0x6D,0x6C,0x20,0x76,0x65,0x72,0x73,0x69,0x6F,0x6E,0x3D,

0x27,0x31,0x2E,0x30,0x27,0x20,0x65,0x6E,0x63,0x6F,0x64,0x69,0x6E,0x67,0x3D,0x27,

0x55,0x54,0x46,0x2D,0x38,0x27,0x20,0x73,0x74,0x61,0x6E,0x64,0x61,0x6C,0x6F,0x6E,

0x65,0x3D,0x27,0x79,0x65,0x73,0x27,0x3F,0x3E,0x0D,0x0A,0x3C,0x61,0x73,0x73,0x65,

0x6D,0x62,0x6C,0x79,0x20,0x78,0x6D,0x6C,0x6E,0x73,0x3D,0x27,0x75,0x72,0x6E,0x3A,

0x73,0x63,0x68,0x65,0x6D,0x61,0x73,0x2D,0x6D,0x69,0x63,0x72,0x6F,0x73,0x6F,0x66,

0x74,0x2D,0x63,0x6F,0x6D,0x3A,0x61,0x73,0x6D,0x2E,0x76,0x31,0x27,0x20,0x6D,0x61,

0x6E,0x69,0x66,0x65,0x73,0x74,0x56,0x65,0x72,0x73,0x69,0x6F,0x6E,0x3D,0x27,0x31,

0x2E,0x30,0x27,0x3E,0x0D,0x0A,0x20,0x20,0x3C,0x74,0x72,0x75,0x73,0x74,0x49,0x6E,

0x66,0x6F,0x20,0x78,0x6D,0x6C,0x6E,0x73,0x3D,0x22,0x75,0x72,0x6E,0x3A,0x73,0x63,

0x68,0x65,0x6D,0x61,0x73,0x2D,0x6D,0x69,0x63,0x72,0x6F,0x73,0x6F,0x66,0x74,0x2D,

0x63,0x6F,0x6D,0x3A,0x61,0x73,0x6D,0x2E,0x76,0x33,0x22,0x3E,0x0D,0x0A,0x20,0x20,

0x20,0x20,0x3C,0x73,0x65,0x63,0x75,0x72,0x69,0x74,0x79,0x3E,0x0D,0x0A,0x20,0x20,

0x20,0x20,0x20,0x20,0x3C,0x72,0x65,0x71,0x75,0x65,0x73,0x74,0x65,0x64,0x50,0x72,

0x69,0x76,0x69,0x6C,0x65,0x67,0x65,0x73,0x3E,0x0D,0x0A,0x20,0x20,0x20,0x20,0x20,

0x20,0x20,0x20,0x3C,0x72,0x65,0x71,0x75,0x65,0x73,0x74,0x65,0x64,0x45,0x78,0x65,

0x63,0x75,0x74,0x69,0x6F,0x6E,0x4C,0x65,0x76,0x65,0x6C,0x20,0x6C,0x65,0x76,0x65,

0x6C,0x3D,0x27,0x61,0x73,0x49,0x6E,0x76,0x6F,0x6B,0x65,0x72,0x27,0x20,0x75,0x69,

0x41,0x63,0x63,0x65,0x73,0x73,0x3D,0x27,0x66,0x61,0x6C,0x73,0x65,0x27,0x20,0x2F,

0x3E,0x0D,0x0A,0x20,0x20,0x20,0x20,0x20,0x20,0x3C,0x2F,0x72,0x65,0x71,0x75,0x65,

0x73,0x74,0x65,0x64,0x50,0x72,0x69,0x76,0x69,0x6C,0x65,0x67,0x65,0x73,0x3E,0x0D,

0x0A,0x20,0x20,0x20,0x20,0x3C,0x2F,0x73,0x65,0x63,0x75,0x72,0x69,0x74,0x79,0x3E,

0x0D,0x0A,0x20,0x20,0x3C,0x2F,0x74,0x72,0x75,0x73,0x74,0x49,0x6E,0x66,0x6F,0x3E,

0x0D,0x0A,0x3C,0x2F,0x61,0x73,0x73,0x65,0x6D,0x62,0x6C,0x79,0x3E,0x0D,0x0A,

};

#define a123\_10\_SIZE 463

#define a123\_ZEROS\_10 36

#define a123\_ZEROS\_11 464

unsigned char a123\_11[] = {

0x10,0x00,0x00,0x20,0x00,0x00,0x00,0x1D,0x30,0x22,0x30,0x34,0x30,0x3A,0x30,0x4A,

0x30,0x51,0x30,0x57,0x30,0x62,0x30,0x68,0x30,0x9F,0x30,0xAB,0x30,0x00,0x00,0x00,

0x30,0x00,0x00,0x10,0x00,0x00,0x00,0xB4,0x30,0xBA,0x30,0xC0,0x30,0xC6,0x30,

};

#define a123\_11\_SIZE 47

unsigned long long int a123\_array\_part\_count = 12;

unsigned char \* a123[12] = {

a123\_0

, a123\_1

, a123\_2

, a123\_3

, a123\_4

, a123\_5

, a123\_6

, a123\_7

, a123\_8

, a123\_9

, a123\_10

, a123\_11

};

unsigned long long int a123\_array\_part\_size[12] = {

a123\_0\_SIZE

, a123\_1\_SIZE

, a123\_2\_SIZE

, a123\_3\_SIZE

, a123\_4\_SIZE

, a123\_5\_SIZE

, a123\_6\_SIZE

, a123\_7\_SIZE

, a123\_8\_SIZE

, a123\_9\_SIZE

, a123\_10\_SIZE

, a123\_11\_SIZE

};

unsigned long long int a123\_zero\_part\_count = 12;

unsigned long long int a123\_zeros[12] = {

a123\_ZEROS\_0

, a123\_ZEROS\_1

, a123\_ZEROS\_2

, a123\_ZEROS\_3

, a123\_ZEROS\_4

, a123\_ZEROS\_5

, a123\_ZEROS\_6

, a123\_ZEROS\_7

, a123\_ZEROS\_8

, a123\_ZEROS\_9

, a123\_ZEROS\_10

, a123\_ZEROS\_11

};  
  
  
mod.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: mod.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeModCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) { // task

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_MOD);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECXMinus4[] = { 0x8B, 0x41, 0xFC };

const unsigned char code\_\_cdq[] = { 0x99 };

const unsigned char code\_\_idiv\_stackTopByECX[] = { 0xF7, 0x39 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_mov\_eax\_edx[] = { 0x8B, 0xC2 };

const unsigned char code\_\_mov\_toAddrFromECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECXMinus4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cdq, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_idiv\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_edx, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_toAddrFromECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192," ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_MOD][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx - 4]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cdq\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " idiv dword ptr [ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, edx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_MOD][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] %= opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
  
mul.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: mul.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeMulCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_MUL);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECXMinus4[] = { 0x8B, 0x41, 0xFC };

const unsigned char code\_\_imul\_stackTopByECX[] = { 0xF7, 0x29 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_mov\_toAddrFromECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECXMinus4, 3);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cdq, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_imul\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_eax, 1);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_dec\_ebp, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_toAddrFromECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_MUL][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx - 4]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;cdq\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " imul dword ptr [ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_MUL][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] \*= opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
not.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: not.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeNotCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_NOT);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_cmp\_eax\_0[] = { 0x83, 0xF8, 0x00 };

const unsigned char code\_\_sete\_al[] = { 0x0F, 0x94, 0xC0 };

const unsigned char code\_\_and\_eax\_1[] = { 0x83, 0xE0, 0x01 };

//

const unsigned char code\_\_mov\_stackTopByECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_eax\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sete\_al, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_1, 3);

//

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_NOT][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp eax, 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sete al\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and eax, 1\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr[ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_NOT][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex] = !opStack[opStackIndex];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
not\_equal.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: not\_equal.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeIsNotEqualCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_NOT\_EQUAL);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_cmp\_stackTopByECX\_eax[] = { 0x39, 0x01 };

const unsigned char code\_\_setne\_al[] = { 0x0F, 0x95, 0xC0 };

const unsigned char code\_\_and\_eax\_1[] = { 0x83, 0xE0, 0x01 };

const unsigned char code\_\_mov\_stackTopByECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_stackTopByECX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_setne\_al, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_1, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_NOT\_EQUAL][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp dword ptr[ecx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " setne al\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and eax, 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr[ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_NOT\_EQUAL][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] = opStack[opStackIndex - 1] != opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
null\_statement.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: null\_statement.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeNullStatementAfterNonContextCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_NULL\_STATEMENT);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;null statement (non-context)\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " //null statement (non-context)\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
objectcodegen\_pattern.cpp  
// Generated sparse arrays

unsigned char o123\_0[] = {

0x4C,0x01,0x05,0x00,0x85,0x62,0x84,0x67,0x86,0x4B,0x00,0x00,0x1F,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x2E,0x74,0x65,0x78,0x74,0x24,0x6D,0x6E,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0xB2,0x20,0x00,0x00,0xDC,0x00,0x00,0x00,0x8E,0x21,0x00,0x00,

0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x20,0x00,0x50,0x60,0x2E,0x64,0x61,0x74,

0x61,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x91,0x21,0x00,0x00,

0x2E,0x22,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x40,0x00,0x50,0xC0,0x2E,0x64,0x65,0x62,0x75,0x67,0x24,0x53,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x4C,0x05,0x00,0x00,0xBF,0x43,0x00,0x00,0x0C,0x49,0x00,0x00,

0x00,0x00,0x00,0x00,0x1E,0x00,0x00,0x00,0x40,0x00,0x10,0x42,0x2E,0x64,0x65,0x62,

0x75,0x67,0x24,0x54,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,0x01,0x00,0x00,

0x38,0x4A,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x40,0x00,0x10,0x42,0x2E,0x64,0x72,0x65,0x63,0x74,0x76,0x65,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x0D,0x00,0x00,0x00,0x78,0x4B,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0A,0x00,0x00,0xE8,0x00,0x00,0x00,

0x00,0x5E,0x83,0xEE,0x05,0x8B,0xFE,0x81,0xC7,0x00,0x40,0x00,0x00,0x8B,0xCF,0x81,

0xC1,0x00,0x02,0x00,0x00,0xEB,0x7C,0x50,0x68,0x00,0x00,0x00,0x00,0x68,0x00,0x00,

0x00,0x00,0xE8,0x00,0x00,0x00,0x00,0x83,0xC4,0x0C,0x6A,0x00,0x6A,0x00,0x50,0x68,

0x00,0x00,0x00,0x00,0xFF,0x35,0x00,0x00,0x00,0x00,0xE8,0x00,0x00,0x00,0x00,0xC3,

0x55,0x8B,0xEC,0x6A,0x00,0x68,0x00,0x00,0x00,0x00,0x6A,0x0F,0x68,0x01,0x00,0x00,

0x00,0xFF,0x35,0x00,0x00,0x00,0x00,0xE8,0x00,0x00,0x00,0x00,0x8D,0x35,0x00,0x00,

0x00,0x00,0x03,0x35,0x00,0x00,0x00,0x00,0x83,0xEE,0x02,0xE8,0x04,0x00,0x00,0x00,

0x8B,0xE5,0x5D,0xC3,0x33,0xC0,0xBB,0x01,0x00,0x00,0x00,0x33,0xC9,0x0F,0xB6,0x0E,

0x85,0xC9,0x74,0x0E,0x83,0xE9,0x30,0x0F,0xAF,0xCB,0x03,0xC1,0x6B,0xDB,0x0A,0x4E,

0xEB,0xEB,0xC3,0x6A,0xF6,0xE8,0x00,0x00,0x00,0x00,0xA3,0x00,0x00,0x00,0x00,0x6A,

0xF5,0xE8,0x00,0x00,0x00,0x00,0xA3,0x00,0x00,0x00,0x00,0x33,0xC0,0xC3,

};

#define o123\_0\_SIZE 398

#define o123\_ZEROS\_0 8192

unsigned char o123\_1[] = {

0x1D,0x00,0x00,0x00,0x14,0x00,0x00,0x00,0x06,0x00,0x22,0x00,0x00,0x00,0x0E,0x00,

0x00,0x00,0x06,0x00,0x27,0x00,0x00,0x00,0x0D,0x00,0x00,0x00,0x14,0x00,0x34,0x00,

0x00,0x00,0x0E,0x00,0x00,0x00,0x06,0x00,0x3A,0x00,0x00,0x00,0x10,0x00,0x00,0x00,

0x06,0x00,0x3F,0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x14,0x00,0x4A,0x00,0x00,0x00,

0x11,0x00,0x00,0x00,0x06,0x00,0x51,0x00,0x00,0x00,0x12,0x00,0x00,0x00,0x06,0x00,

0x57,0x00,0x00,0x00,0x0F,0x00,0x00,0x00,0x06,0x00,0x5C,0x00,0x00,0x00,0x0B,0x00,

0x00,0x00,0x14,0x00,0x62,0x00,0x00,0x00,0x12,0x00,0x00,0x00,0x06,0x00,0x68,0x00,

0x00,0x00,0x11,0x00,0x00,0x00,0x06,0x00,0x9A,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,

0x14,0x00,0x9F,0x00,0x00,0x00,0x0F,0x00,0x00,0x00,0x06,0x00,0xA6,0x00,0x00,0x00,

0x0A,0x00,0x00,0x00,0x14,0x00,0xAB,0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x06,

};

#define o123\_1\_SIZE 159

#define o123\_ZEROS\_1 8449

unsigned char o123\_2[] = {

0x25,0x64,0x0A,0x0D,

};

#define o123\_2\_SIZE 4

#define o123\_ZEROS\_2 141

unsigned char o123\_3[] = {

0x04,0x00,0x00,0x00,0xF3,0x00,0x00,0x00,0x3C,0x00,0x00,0x00,0x00,0x43,0x3A,0x5C,

0x55,0x73,0x65,0x72,0x73,0x5C,0x4E,0x61,0x7A,0x61,0x72,0x5C,0x73,0x6F,0x75,0x72,

0x63,0x65,0x5C,0x72,0x65,0x70,0x6F,0x73,0x5C,0x50,0x72,0x6F,0x6A,0x65,0x63,0x74,

0x36,0x30,0x5C,0x50,0x72,0x6F,0x6A,0x65,0x63,0x74,0x36,0x30,0x5C,0x73,0x6F,0x75,

0x72,0x63,0x65,0x2E,0x61,0x73,0x6D,0x00,0xF4,0x00,0x00,0x00,0x18,0x00,0x00,0x00,

0x01,0x00,0x00,0x00,0x10,0x01,0xD3,0x4E,0x23,0x32,0x9C,0x94,0x3A,0xC3,0x61,0x14,

0xF6,0x0B,0x9E,0xC4,0xCE,0x14,0x00,0x00,0xF2,0x00,0x00,0x00,0xE8,0x01,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xB2,0x20,0x00,0x00,0x00,0x00,0x00,0x00,

0x3A,0x00,0x00,0x00,0xDC,0x01,0x00,0x00,0x05,0x00,0x00,0x00,0x23,0x00,0x00,0x80,

0x06,0x00,0x00,0x00,0x24,0x00,0x00,0x80,0x09,0x00,0x00,0x00,0x25,0x00,0x00,0x80,

0x0B,0x00,0x00,0x00,0x26,0x00,0x00,0x80,0x11,0x00,0x00,0x00,0x27,0x00,0x00,0x80,

0x13,0x00,0x00,0x00,0x28,0x00,0x00,0x80,0x19,0x00,0x00,0x00,0x29,0x00,0x00,0x80,

0x1B,0x00,0x00,0x00,0x2A,0x00,0x00,0x80,0x1B,0x00,0x00,0x00,0x2B,0x00,0x00,0x80,

0x1C,0x00,0x00,0x00,0x2C,0x00,0x00,0x80,0x21,0x00,0x00,0x00,0x2D,0x00,0x00,0x80,

0x26,0x00,0x00,0x00,0x2E,0x00,0x00,0x80,0x2B,0x00,0x00,0x00,0x2F,0x00,0x00,0x80,

0x2E,0x00,0x00,0x00,0x37,0x00,0x00,0x80,0x30,0x00,0x00,0x00,0x38,0x00,0x00,0x80,

0x32,0x00,0x00,0x00,0x39,0x00,0x00,0x80,0x33,0x00,0x00,0x00,0x3A,0x00,0x00,0x80,

0x38,0x00,0x00,0x00,0x3B,0x00,0x00,0x80,0x3E,0x00,0x00,0x00,0x3C,0x00,0x00,0x80,

0x43,0x00,0x00,0x00,0x3E,0x00,0x00,0x80,0x44,0x00,0x00,0x00,0x42,0x00,0x00,0x80,

0x44,0x00,0x00,0x00,0x43,0x00,0x00,0x80,0x45,0x00,0x00,0x00,0x44,0x00,0x00,0x80,

0x47,0x00,0x00,0x00,0x46,0x00,0x00,0x80,0x49,0x00,0x00,0x00,0x47,0x00,0x00,0x80,

0x4E,0x00,0x00,0x00,0x48,0x00,0x00,0x80,0x50,0x00,0x00,0x00,0x49,0x00,0x00,0x80,

0x55,0x00,0x00,0x00,0x4A,0x00,0x00,0x80,0x5B,0x00,0x00,0x00,0x4B,0x00,0x00,0x80,

0x60,0x00,0x00,0x00,0x4D,0x00,0x00,0x80,0x66,0x00,0x00,0x00,0x4E,0x00,0x00,0x80,

0x6C,0x00,0x00,0x00,0x4F,0x00,0x00,0x80,0x6F,0x00,0x00,0x00,0x50,0x00,0x00,0x80,

0x74,0x00,0x00,0x00,0x52,0x00,0x00,0x80,0x76,0x00,0x00,0x00,0x53,0x00,0x00,0x80,

0x77,0x00,0x00,0x00,0x54,0x00,0x00,0x80,0x78,0x00,0x00,0x00,0x57,0x00,0x00,0x80,

0x78,0x00,0x00,0x00,0x5A,0x00,0x00,0x80,0x7A,0x00,0x00,0x00,0x5B,0x00,0x00,0x80,

0x7F,0x00,0x00,0x00,0x5C,0x00,0x00,0x80,0x81,0x00,0x00,0x00,0x5F,0x00,0x00,0x80,

0x84,0x00,0x00,0x00,0x60,0x00,0x00,0x80,0x86,0x00,0x00,0x00,0x61,0x00,0x00,0x80,

0x88,0x00,0x00,0x00,0x62,0x00,0x00,0x80,0x8B,0x00,0x00,0x00,0x63,0x00,0x00,0x80,

0x8E,0x00,0x00,0x00,0x64,0x00,0x00,0x80,0x90,0x00,0x00,0x00,0x65,0x00,0x00,0x80,

0x93,0x00,0x00,0x00,0x66,0x00,0x00,0x80,0x94,0x00,0x00,0x00,0x67,0x00,0x00,0x80,

0x96,0x00,0x00,0x00,0x6A,0x00,0x00,0x80,0x97,0x00,0x00,0x00,0x6E,0x00,0x00,0x80,

0x99,0x00,0x00,0x00,0x6F,0x00,0x00,0x80,0x9E,0x00,0x00,0x00,0x70,0x00,0x00,0x80,

0xA3,0x00,0x00,0x00,0x71,0x00,0x00,0x80,0xA5,0x00,0x00,0x00,0x72,0x00,0x00,0x80,

0xAA,0x00,0x00,0x00,0x73,0x00,0x00,0x80,0xAF,0x00,0x00,0x00,0x87,0x00,0x00,0x80,

0xB1,0x00,0x00,0x00,0x88,0x00,0x00,0x80,0xF1,0x00,0x00,0x00,0xEA,0x02,0x00,0x00,

0x49,0x00,0x01,0x11,0x00,0x00,0x00,0x00,0x43,0x3A,0x5C,0x55,0x73,0x65,0x72,0x73,

0x5C,0x4E,0x61,0x7A,0x61,0x72,0x5C,0x73,0x6F,0x75,0x72,0x63,0x65,0x5C,0x72,0x65,

0x70,0x6F,0x73,0x5C,0x50,0x72,0x6F,0x6A,0x65,0x63,0x74,0x36,0x30,0x5C,0x50,0x72,

0x6F,0x6A,0x65,0x63,0x74,0x36,0x30,0x5C,0x52,0x65,0x6C,0x65,0x61,0x73,0x65,0x5C,

0x73,0x6F,0x75,0x72,0x63,0x65,0x2E,0x6F,0x62,0x6A,0x00,0x37,0x00,0x3C,0x11,0x03,

0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x00,0x1C,

0x00,0xE4,0x74,0x00,0x00,0x4D,0x69,0x63,0x72,0x6F,0x73,0x6F,0x66,0x74,0x20,0x28,

0x52,0x29,0x20,0x4D,0x61,0x63,0x72,0x6F,0x20,0x41,0x73,0x73,0x65,0x6D,0x62,0x6C,

0x65,0x72,0x00,0x00,0xBB,0x00,0x3D,0x11,0x00,0x63,0x77,0x64,0x00,0x43,0x3A,0x5C,

0x55,0x73,0x65,0x72,0x73,0x5C,0x4E,0x61,0x7A,0x61,0x72,0x5C,0x73,0x6F,0x75,0x72,

0x63,0x65,0x5C,0x72,0x65,0x70,0x6F,0x73,0x5C,0x50,0x72,0x6F,0x6A,0x65,0x63,0x74,

0x36,0x30,0x5C,0x50,0x72,0x6F,0x6A,0x65,0x63,0x74,0x36,0x30,0x00,0x65,0x78,0x65,

0x00,0x43,0x3A,0x5C,0x50,0x72,0x6F,0x67,0x72,0x61,0x6D,0x20,0x46,0x69,0x6C,0x65,

0x73,0x20,0x28,0x78,0x38,0x36,0x29,0x5C,0x4D,0x69,0x63,0x72,0x6F,0x73,0x6F,0x66,

0x74,0x20,0x56,0x69,0x73,0x75,0x61,0x6C,0x20,0x53,0x74,0x75,0x64,0x69,0x6F,0x5C,

0x32,0x30,0x31,0x39,0x5C,0x45,0x6E,0x74,0x65,0x72,0x70,0x72,0x69,0x73,0x65,0x5C,

0x56,0x43,0x5C,0x54,0x6F,0x6F,0x6C,0x73,0x5C,0x4D,0x53,0x56,0x43,0x5C,0x31,0x34,

0x2E,0x32,0x38,0x2E,0x32,0x39,0x39,0x31,0x30,0x5C,0x62,0x69,0x6E,0x5C,0x48,0x6F,

0x73,0x74,0x58,0x38,0x36,0x5C,0x78,0x38,0x36,0x5C,0x6D,0x6C,0x2E,0x65,0x78,0x65,

0x00,0x73,0x72,0x63,0x00,0x73,0x6F,0x75,0x72,0x63,0x65,0x2E,0x61,0x73,0x6D,0x00,

0x00,0x1A,0x00,0x0C,0x11,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x76,

0x61,0x6C,0x75,0x65,0x54,0x65,0x6D,0x70,0x5F,0x6D,0x73,0x67,0x00,0x1A,0x00,0x0C,

0x11,0x22,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x68,0x43,0x6F,0x6E,0x73,

0x6F,0x6C,0x65,0x49,0x6E,0x70,0x75,0x74,0x00,0x1B,0x00,0x0C,0x11,0x22,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x68,0x43,0x6F,0x6E,0x73,0x6F,0x6C,0x65,0x4F,

0x75,0x74,0x70,0x75,0x74,0x00,0x2D,0x00,0x10,0x11,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x34,0x00,

0x00,0x00,0x06,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x67,0x65,0x74,

0x50,0x72,0x6F,0x63,0x00,0x02,0x00,0x06,0x00,0x2D,0x00,0x10,0x11,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x29,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x29,0x00,0x00,0x00,0x08,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x70,0x75,0x74,0x50,0x72,0x6F,0x63,0x00,0x02,0x00,0x06,0x00,0x15,0x00,0x05,0x11,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x69,0x6E,0x69,0x74,0x43,0x6F,0x6E,0x73,0x6F,

0x6C,0x65,0x00,0x19,0x00,0x0C,0x11,0x22,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x72,0x65,0x61,0x64,0x4F,0x75,0x74,0x43,0x6F,0x75,0x6E,0x74,0x00,0x13,0x00,

0x0C,0x11,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x62,0x75,0x66,0x66,

0x65,0x72,0x00,0x0F,0x00,0x05,0x11,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x73,0x74,

0x61,0x72,0x74,0x00,0x33,0x00,0x10,0x11,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x1F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0x00,0x00,0x00,

0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x73,0x74,0x72,0x69,0x6E,

0x67,0x5F,0x74,0x6F,0x5F,0x69,0x6E,0x74,0x00,0x16,0x00,0x05,0x11,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x63,0x6F,0x6E,0x76,0x65,0x72,0x74,0x5F,0x6C,0x6F,0x6F,0x70,

0x00,0x0E,0x00,0x05,0x11,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x64,0x6F,0x6E,0x65,

0x00,0x02,0x00,0x06,0x00,0x17,0x00,0x0C,0x11,0x20,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x64,0x61,0x74,0x61,0x5F,0x73,0x74,0x61,0x72,0x74,0x00,0x1A,0x00,

0x0C,0x11,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x76,0x61,0x6C,0x75,

0x65,0x54,0x65,0x6D,0x70,0x5F,0x66,0x6D,0x74,0x00,0x00,0x00,0x00,0x70,0x00,0x00,

0x00,0x18,0x00,0x00,0x00,0x0B,0x00,0x74,0x00,0x00,0x00,0x18,0x00,0x00,0x00,0x0A,

0x00,0xA9,0x03,0x00,0x00,0x0E,0x00,0x00,0x00,0x0B,0x00,0xAD,0x03,0x00,0x00,0x0E,

0x00,0x00,0x00,0x0A,0x00,0xC5,0x03,0x00,0x00,0x0F,0x00,0x00,0x00,0x0B,0x00,0xC9,

0x03,0x00,0x00,0x0F,0x00,0x00,0x00,0x0A,0x00,0xE1,0x03,0x00,0x00,0x10,0x00,0x00,

0x00,0x0B,0x00,0xE5,0x03,0x00,0x00,0x10,0x00,0x00,0x00,0x0A,0x00,0x16,0x04,0x00,

0x00,0x16,0x00,0x00,0x00,0x0B,0x00,0x1A,0x04,0x00,0x00,0x16,0x00,0x00,0x00,0x0A,

0x00,0x49,0x04,0x00,0x00,0x15,0x00,0x00,0x00,0x0B,0x00,0x4D,0x04,0x00,0x00,0x15,

0x00,0x00,0x00,0x0A,0x00,0x60,0x04,0x00,0x00,0x19,0x00,0x00,0x00,0x0B,0x00,0x64,

0x04,0x00,0x00,0x19,0x00,0x00,0x00,0x0A,0x00,0x7B,0x04,0x00,0x00,0x11,0x00,0x00,

0x00,0x0B,0x00,0x7F,0x04,0x00,0x00,0x11,0x00,0x00,0x00,0x0A,0x00,0x96,0x04,0x00,

0x00,0x12,0x00,0x00,0x00,0x0B,0x00,0x9A,0x04,0x00,0x00,0x12,0x00,0x00,0x00,0x0A,

0x00,0xA7,0x04,0x00,0x00,0x13,0x00,0x00,0x00,0x0B,0x00,0xAB,0x04,0x00,0x00,0x13,

0x00,0x00,0x00,0x0A,0x00,0xD4,0x04,0x00,0x00,0x17,0x00,0x00,0x00,0x0B,0x00,0xD8,

0x04,0x00,0x00,0x17,0x00,0x00,0x00,0x0A,0x00,0xED,0x04,0x00,0x00,0x1A,0x00,0x00,

0x00,0x0B,0x00,0xF1,0x04,0x00,0x00,0x1A,0x00,0x00,0x00,0x0A,0x00,0x05,0x05,0x00,

0x00,0x1B,0x00,0x00,0x00,0x0B,0x00,0x09,0x05,0x00,0x00,0x1B,0x00,0x00,0x00,0x0A,

0x00,0x1D,0x05,0x00,0x00,0x1C,0x00,0x00,0x00,0x0B,0x00,0x21,0x05,0x00,0x00,0x1C,

0x00,0x00,0x00,0x0A,0x00,0x36,0x05,0x00,0x00,0x14,0x00,0x00,0x00,0x0B,0x00,0x3A,

0x05,0x00,0x00,0x14,0x00,0x00,0x00,0x0A,0x00,0x04,0x00,0x00,0x00,0x0A,0x00,0x01,

0x12,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x00,0x08,0x10,0x03,0x00,0x00,

0x00,0x00,0x00,0x01,0x00,0x00,0x10,0x00,0x00,0x06,0x00,0x0E,0x00,0x00,0x00,0xF2,

0xF1,0x1A,0x00,0x01,0x12,0x05,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,

0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x0E,0x00,0x08,

0x10,0x03,0x00,0x00,0x00,0x07,0x00,0x05,0x00,0x03,0x10,0x00,0x00,0x06,0x00,0x01,

0x12,0x00,0x00,0x00,0x00,0x0E,0x00,0x08,0x10,0x03,0x00,0x00,0x00,0x07,0x00,0x00,

0x00,0x05,0x10,0x00,0x00,0x06,0x00,0x01,0x12,0x00,0x00,0x00,0x00,0x0E,0x00,0x08,

0x10,0x03,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x07,0x10,0x00,0x00,0x0A,0x00,0x01,

0x12,0x01,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x0E,0x00,0x08,0x10,0x03,0x00,0x00,

0x00,0x07,0x00,0x01,0x00,0x09,0x10,0x00,0x00,0x0E,0x00,0x01,0x12,0x02,0x00,0x00,

0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x0E,0x00,0x08,0x10,0x03,0x00,0x00,

0x00,0x07,0x00,0x02,0x00,0x0B,0x10,0x00,0x00,0x0E,0x00,0x01,0x12,0x02,0x00,0x00,

0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x0E,0x00,0x08,0x10,0x03,0x00,0x00,

0x00,0x07,0x00,0x02,0x00,0x0D,0x10,0x00,0x00,0x06,0x00,0x01,0x12,0x00,0x00,0x00,

0x00,0x0E,0x00,0x08,0x10,0x03,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x0F,0x10,0x00,

0x00,0x1A,0x00,0x01,0x12,0x05,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,

0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x0E,0x00,0x08,

0x10,0x03,0x00,0x00,0x00,0x07,0x00,0x05,0x00,0x11,0x10,0x00,0x00,0x0A,0x00,0x01,

0x12,0x01,0x00,0x00,0x00,0x22,0x00,0x00,0x00,0x0E,0x00,0x08,0x10,0x03,0x00,0x00,

0x00,0x07,0x00,0x01,0x00,0x13,0x10,0x00,0x00,0x2F,0x45,0x4E,0x54,0x52,0x59,0x3A,

0x73,0x74,0x61,0x72,0x74,0x20,0x00,0x40,0x63,0x6F,0x6D,0x70,0x2E,0x69,0x64,0xE4,

0x74,0x03,0x01,0xFF,0xFF,0x00,0x00,0x03,0x00,0x40,0x66,0x65,0x61,0x74,0x2E,0x30,

0x30,0x10,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,0x03,0x00,0x2E,0x74,0x65,0x78,0x74,

0x24,0x6D,0x6E,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x03,0x01,0xB2,0x20,0x00,

0x00,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x2E,

0x64,0x61,0x74,0x61,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03,

0x01,0x91,0x21,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x2E,0x64,0x65,0x62,0x75,0x67,0x24,0x53,0x00,0x00,0x00,0x00,0x03,

0x00,0x00,0x00,0x03,0x01,0x4C,0x05,0x00,0x00,0x1E,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x2E,0x64,0x65,0x62,0x75,0x67,0x24,0x54,0x00,

0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x03,0x01,0x40,0x01,

};

#define o123\_3\_SIZE 2155

#define o123\_ZEROS\_3 20

unsigned char o123\_4[] = {

0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x02,0x00,0x00,0x00,

0x00,0x00,0x14,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x02,0x00,

0x00,0x00,0x00,0x00,0x25,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x00,

0x02,0x00,0x00,0x00,0x00,0x00,0x37,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x20,0x00,0x02,0x00,0x00,0x00,0x00,0x00,0x42,0x00,0x00,0x00,0x00,0x20,0x00,0x00,

0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x50,0x00,0x00,0x00,0x05,0x21,

0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x5E,0x00,0x00,0x00,

0x09,0x21,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x6D,0x00,

0x00,0x00,0x8D,0x21,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x62,0x75,0x66,0x66,

0x65,0x72,0x00,0x00,0x0D,0x21,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x5F,0x73,

0x74,0x61,0x72,0x74,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,

0x00,0x00,0x00,0x00,0x7A,0x00,0x00,0x00,0x00,0x21,0x00,0x00,0x02,0x00,0x00,0x00,

0x03,0x00,0x00,0x00,0x00,0x00,0x88,0x00,0x00,0x00,0x1B,0x00,0x00,0x00,0x01,0x00,

0x20,0x00,0x02,0x00,0x00,0x00,0x00,0x00,0x93,0x00,0x00,0x00,0x44,0x00,0x00,0x00,

0x01,0x00,0x20,0x00,0x02,0x00,0x00,0x00,0x00,0x00,0x9E,0x00,0x00,0x00,0x78,0x00,

0x00,0x00,0x01,0x00,0x20,0x00,0x02,0x00,0x24,0x24,0x30,0x30,0x30,0x30,0x30,0x30,

0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0xAF,0x00,

0x00,0x00,0x97,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,

0xBB,0x00,0x00,0x00,0x81,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x06,0x00,0x64,0x6F,

0x6E,0x65,0x00,0x00,0x00,0x00,0x96,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x06,0x00,

0x00,0x00,0x00,0x00,0xC8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x02,0x00,0x00,0x00,

0x03,0x00,0x2E,0x64,0x72,0x65,0x63,0x74,0x76,0x65,0x00,0x00,0x00,0x00,0x05,0x00,

0x00,0x00,0x03,0x01,0x0D,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0xD3,0x00,0x00,0x00,0x5F,0x47,0x65,0x74,0x53,0x74,

0x64,0x48,0x61,0x6E,0x64,0x6C,0x65,0x40,0x34,0x00,0x5F,0x52,0x65,0x61,0x64,0x43,

0x6F,0x6E,0x73,0x6F,0x6C,0x65,0x41,0x40,0x32,0x30,0x00,0x5F,0x57,0x72,0x69,0x74,

0x65,0x43,0x6F,0x6E,0x73,0x6F,0x6C,0x65,0x41,0x40,0x32,0x30,0x00,0x5F,0x77,0x73,

0x70,0x72,0x69,0x6E,0x74,0x66,0x41,0x00,0x76,0x61,0x6C,0x75,0x65,0x54,0x65,0x6D,

0x70,0x5F,0x6D,0x73,0x67,0x00,0x68,0x43,0x6F,0x6E,0x73,0x6F,0x6C,0x65,0x49,0x6E,

0x70,0x75,0x74,0x00,0x68,0x43,0x6F,0x6E,0x73,0x6F,0x6C,0x65,0x4F,0x75,0x74,0x70,

0x75,0x74,0x00,0x72,0x65,0x61,0x64,0x4F,0x75,0x74,0x43,0x6F,0x75,0x6E,0x74,0x00,

0x76,0x61,0x6C,0x75,0x65,0x54,0x65,0x6D,0x70,0x5F,0x66,0x6D,0x74,0x00,0x5F,0x70,

0x75,0x74,0x50,0x72,0x6F,0x63,0x40,0x30,0x00,0x5F,0x67,0x65,0x74,0x50,0x72,0x6F,

0x63,0x40,0x30,0x00,0x5F,0x73,0x74,0x72,0x69,0x6E,0x67,0x5F,0x74,0x6F,0x5F,0x69,

0x6E,0x74,0x40,0x30,0x00,0x69,0x6E,0x69,0x74,0x43,0x6F,0x6E,0x73,0x6F,0x6C,0x65,

0x00,0x63,0x6F,0x6E,0x76,0x65,0x72,0x74,0x5F,0x6C,0x6F,0x6F,0x70,0x00,0x64,0x61,

0x74,0x61,0x5F,0x73,0x74,0x61,0x72,0x74,0x00,

};

#define o123\_4\_SIZE 585

unsigned long long int o123\_array\_part\_count = 5;

unsigned char\* o123[5] = {

o123\_0

, o123\_1

, o123\_2

, o123\_3

, o123\_4

};

unsigned long long int o123\_array\_part\_size[5] = {

o123\_0\_SIZE

, o123\_1\_SIZE

, o123\_2\_SIZE

, o123\_3\_SIZE

, o123\_4\_SIZE

};

unsigned long long int o123\_zero\_part\_count = 4;

unsigned long long int o123\_zeros[4] = {

o123\_ZEROS\_0

, o123\_ZEROS\_1

, o123\_ZEROS\_2

, o123\_ZEROS\_3

};  
  
operand.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: identifier\_or\_value.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

unsigned char\* makeValueCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if ((\*lastLexemInfoInTable)->tokenType == VALUE\_LEXEME\_TYPE) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_add\_ecx\_4[] = { 0x83, 0xC1, 0x04 };

const unsigned char code\_\_mov\_eax\_value[] = { 0xB8, 0x00, 0x00, 0x00, 0x00 };

unsigned char code\_\_mov\_toAddrFromECX\_eax[] = { 0x89, 0x01 };

// const unsigned char\* valueParts = (const unsigned char\*)&(\*lastLexemInfoInTable)->ifvalue;

// code\_\_mov\_toAddrFromECX\_value[2] = valueParts[0];

// code\_\_mov\_toAddrFromECX\_value[3] = valueParts[1];

// code\_\_mov\_toAddrFromECX\_value[4] = valueParts[2];

// code\_\_mov\_toAddrFromECX\_value[5] = valueParts[3];

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_value, 5);

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)(\*lastLexemInfoInTable)->ifvalue;

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_toAddrFromECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%lld\"\r\n", (\*lastLexemInfoInTable)->ifvalue);

//

currBytePtr += sprintf((char\*)currBytePtr, " add ecx, 4\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " mov eax, 0%08Xh\r\n", (int)(\*lastLexemInfoInTable)->ifvalue);

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%lld\"\r\n", (\*lastLexemInfoInTable)->ifvalue);

//

currBytePtr += sprintf((char\*)currBytePtr, " opStack[++opStackIndex] = opTemp = 0x%08X;\r\n", (int)(\*lastLexemInfoInTable)->ifvalue);

}

return ++ \* lastLexemInfoInTable, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeIdentifierCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

if ((\*lastLexemInfoInTable)->tokenType == IDENTIFIER\_LEXEME\_TYPE) {

bool findComplete = false;

unsigned long long int variableIndex = 0;

for (; identifierIdsTable[variableIndex][0] != '\0'; ++variableIndex) {

if (!strncmp((\*lastLexemInfoInTable)->lexemStr, identifierIdsTable[variableIndex], MAX\_LEXEM\_SIZE)) {

findComplete = true;

break;

}

}

if (!findComplete) {

printf("\r\nError!\r\n");

exit(0);

}

variableIndex \*= VALUE\_SIZE;

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

unsigned char code\_\_mov\_eax\_edi[] = { 0x8B, 0xC7 };

unsigned char code\_\_add\_eax\_variableOffsetInDataSection[] = { 0x05, 0x00, 0x00, 0x00, 0x00 };

const unsigned char code\_\_mov\_eax\_valueByAdrressInEAX[] = { 0x8B, 0x00 };

const unsigned char code\_\_add\_ecx\_4[] = { 0x83, 0xC1, 0x04 };

const unsigned char code\_\_mov\_toAddrFromECX\_eax[] = { 0x89, 0x01 };

const unsigned char\* variableIndexValueParts = (const unsigned char\*)&variableIndex;

code\_\_add\_eax\_variableOffsetInDataSection[1] = variableIndexValueParts[0];

code\_\_add\_eax\_variableOffsetInDataSection[2] = variableIndexValueParts[1];

code\_\_add\_eax\_variableOffsetInDataSection[3] = variableIndexValueParts[2];

code\_\_add\_eax\_variableOffsetInDataSection[4] = variableIndexValueParts[3];

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_eax\_variableOffsetInDataSection, 5);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_valueByAdrressInEAX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_toAddrFromECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", (\*lastLexemInfoInTable)->lexemStr);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, edi\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " add eax, 0%08Xh\r\n", (int)variableIndex);

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[eax]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", (\*lastLexemInfoInTable)->lexemStr);

//

currBytePtr += sprintf((char\*)currBytePtr, " opStack[++opStackIndex] = opTemp = data[0x%08X];\r\n", (int)variableIndex);

}

return ++ \* lastLexemInfoInTable, currBytePtr;

}

return currBytePtr;

}  
  
or.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: or.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeOrCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_OR);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_cmp\_eax\_0[] = { 0x83, 0xF8, 0x00 };

const unsigned char code\_\_setne\_al[] = { 0x0F, 0x95, 0xC0 };

const unsigned char code\_\_and\_eax\_1[] = { 0x83, 0xE0, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

//

const unsigned char code\_\_cmp\_stackTopByECX\_0[] = { 0x83, 0x39, 0x00 };

const unsigned char code\_\_setne\_dl[] = { 0x0F, 0x95, 0xC2 };

const unsigned char code\_\_and\_edx\_1[] = { 0x83, 0xE2, 0x01 };

//

const unsigned char code\_\_or\_eax\_edx[] = { 0x0B, 0xC2 };

//

const unsigned char code\_\_mov\_stackTopByECX\_eax[] = { 0x89, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_eax\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_setne\_al, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_eax\_1, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

//

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_stackTopByECX\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_setne\_dl, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_and\_edx\_1, 3);

//

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_or\_eax\_edx, 2);

//

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_stackTopByECX\_eax, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_OR][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " cmp eax, 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " setne al\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and eax, 1\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " cmp dword ptr[ecx], 0\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " setne dl\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " and edx, 1\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " or eax, edx\r\n");

//

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr[ecx], eax\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_OR][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] = opStack[opStackIndex - 1] || opStack[opStackIndex--];\r\n");

// currBytePtr += sprintf((char\*)currBytePtr, " //--opStackIndex;\r\n");

// currBytePtr += sprintf((char\*)currBytePtr, " //opTemp = opStack[opStackIndex];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
  
output.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: output.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makePutCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_OUTPUT);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_mov\_edx\_address[] = { 0xBA, 0x00, 0x00, 0x00, 0x00 };

const unsigned char code\_\_add\_edx\_esi[] = { 0x03, 0xD6 };

//const unsigned char code\_\_push\_ecx[] = { 0x51 };

//const unsigned char code\_\_push\_ebx[] = { 0x53 };

const unsigned char code\_\_push\_esi[] = { 0x56 };

const unsigned char code\_\_push\_edi[] = { 0x57 };

const unsigned char code\_\_call\_edx[] = { 0xFF, 0xD2 };

const unsigned char code\_\_pop\_edi[] = { 0x5F };

const unsigned char code\_\_pop\_esi[] = { 0x5E };

//const unsigned char code\_\_pop\_ebx[] = { 0x5B };

//const unsigned char code\_\_pop\_ecx[] = { 0x59 };

const unsigned char code\_\_mov\_ecx\_edi[] = { 0x8B, 0xCF };

const unsigned char code\_\_add\_ecx\_512[] = { 0x81, 0xC1, 0x00, 0x02, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_edx\_address, 5);

\*(unsigned int\*)&(currBytePtr[-4]) = (unsigned int)putProcOffset;

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_edx\_esi, 2);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_ecx, 1);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_ebx, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_esi, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_push\_edi, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_call\_edx, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_edi, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_esi, 1);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_ebx, 1);

//currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_pop\_ecx, 1);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ecx\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ecx\_512, 6);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 9192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_OUTPUT][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " mov edx, 0%08Xh\r\n", (unsigned int)putProcOffset);

currBytePtr += sprintf((char\*)currBytePtr, " add edx, esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;push ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " push edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " call edx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " pop edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " pop esi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;pop ebx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " ;pop ecx\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ecx, edi ; reset second stack\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ecx, 512 ; reset second stack\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_OUTPUT][0]);

currBytePtr += sprintf((char\*)currBytePtr, " (void)printf(\"%%d\\r\\n\", opTemp = opStack[opStackIndex]), opStackIndex = 0;\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
preparer.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: preparer.hxx \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../../src/include/preparer/preparer.h"

#include "../../../src/include/def.h"

#include "../../../src/include/config.h"

#include "../../../src/include/lexica/lexica.h"

#include "../../../src/include/syntax/syntax.h"

#include "../../../src/include/semantix/semantix.h"

#include "../../../src/include/generator/generator.h"

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

int precedenceLevel(char\* lexemStr) {

if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_BITWISE\_NOT][0], MAX\_LEXEM\_SIZE)) {

return 6;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_NOT][0], MAX\_LEXEM\_SIZE)) {

return 6;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_BITWISE\_AND][0], MAX\_LEXEM\_SIZE)) {

return 5;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_AND][0], MAX\_LEXEM\_SIZE)) {

return 5;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_MUL][0], MAX\_LEXEM\_SIZE)) {

return 5;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_DIV][0], MAX\_LEXEM\_SIZE)) {

return 5;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_MOD][0], MAX\_LEXEM\_SIZE)) {

return 5;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_BITWISE\_OR][0], MAX\_LEXEM\_SIZE)) {

return 4;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_OR][0], MAX\_LEXEM\_SIZE)) {

return 4;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_ADD][0], MAX\_LEXEM\_SIZE)) {

return 4;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_SUB][0], MAX\_LEXEM\_SIZE)) {

return 4;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_EQUAL][0], MAX\_LEXEM\_SIZE)) {

return 3;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_NOT\_EQUAL][0], MAX\_LEXEM\_SIZE)) {

return 3;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_LESS][0], MAX\_LEXEM\_SIZE)) {

return 3;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_GREATER][0], MAX\_LEXEM\_SIZE)) {

return 3;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_LESS\_OR\_EQUAL][0], MAX\_LEXEM\_SIZE)) {

return 3;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_RLBIND][0], MAX\_LEXEM\_SIZE)) {

return 2;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_LRBIND][0], MAX\_LEXEM\_SIZE)) {

return 2;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)) {

return 1;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_OUTPUT][0], MAX\_LEXEM\_SIZE)) {

return 1;

}

return 0;

}

bool isLeftAssociative(char\* lexemStr) {

if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_BITWISE\_AND][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_AND][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_MUL][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_DIV][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_MOD][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_BITWISE\_OR][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_OR][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_ADD][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_SUB][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_EQUAL][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_NOT\_EQUAL][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_LESS][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_GREATER][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_LESS\_OR\_EQUAL][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_LRBIND][0], MAX\_LEXEM\_SIZE)) { // ! TODO: ...

return false;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_RLBIND][0], MAX\_LEXEM\_SIZE)) {

return false;

}

if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_BITWISE\_NOT][0], MAX\_LEXEM\_SIZE)) {

return false;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_NOT][0], MAX\_LEXEM\_SIZE)) {

return false;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)) {

return false;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_OUTPUT][0], MAX\_LEXEM\_SIZE)) {

return false;

}

return false;

}

bool isSplittingOperator(char\* lexemStr) {

if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)) {

return true;

}

else if (!strncmp(lexemStr, tokenStruct[MULTI\_TOKEN\_OUTPUT][0], MAX\_LEXEM\_SIZE)) {

return true;

}

return false;

}

void makePrepare4IdentifierOrValue(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable) { //

if ((\*lastLexemInfoInTable)->tokenType == IDENTIFIER\_LEXEME\_TYPE || (\*lastLexemInfoInTable)->tokenType == VALUE\_LEXEME\_TYPE) {

int prevNonOpenParenthesesIndex = -1;

for (; !strncmp((\*lastLexemInfoInTable)[prevNonOpenParenthesesIndex].lexemStr, "(", MAX\_LEXEM\_SIZE); --prevNonOpenParenthesesIndex);

if (!strncmp((\*lastLexemInfoInTable)[1].lexemStr, tokenStruct[MULTI\_TOKEN\_RLBIND][0], MAX\_LEXEM\_SIZE)

||

!strncmp((\*lastLexemInfoInTable)[-1].lexemStr, tokenStruct[MULTI\_TOKEN\_LRBIND][0], MAX\_LEXEM\_SIZE)

||

//!strncmp((\*lastLexemInfoInTable)[-1].lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)

//||

//!strncmp((\*lastLexemInfoInTable)[-2].lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)

//||

!strncmp((\*lastLexemInfoInTable)[prevNonOpenParenthesesIndex].lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)

) {

bool findComplete = false;

for (unsigned long long int index = 0; identifierIdsTable[index][0] != '\0'; ++index) {

if (!strncmp((\*lastLexemInfoInTable)->lexemStr, identifierIdsTable[index], MAX\_LEXEM\_SIZE)) {

findComplete = true;

(\*lastTempLexemInfoInTable)->ifvalue = /\*dataOffset + \*/VALUE\_SIZE \* /\*(unsigned long long int)\*/index;

\_itoa((\*lastTempLexemInfoInTable)->ifvalue, (\*lastTempLexemInfoInTable)->lexemStr, 10);

((\*lastTempLexemInfoInTable)++)->tokenType = VALUE\_LEXEME\_TYPE; // ADDRESS\_LEXEME\_TYPE

++\* lastLexemInfoInTable;

}

}

if (!findComplete) {

printf("\r\nError!\r\n");

exit(0);

}

}

else {

\*(\*lastTempLexemInfoInTable)++ = \*(\*lastLexemInfoInTable)++;

}

}

}

void makePrepare4Operators(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable) {

if (precedenceLevel((\*lastLexemInfoInTable)->lexemStr)) {

while (lexemInfoTransformationTempStackSize > 0) {

struct LexemInfo/\*&\*/ currLexemInfo = lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1];

if (precedenceLevel(currLexemInfo.lexemStr) && (

(isLeftAssociative((\*lastLexemInfoInTable)->lexemStr) && (precedenceLevel((\*lastLexemInfoInTable)->lexemStr) <= precedenceLevel(currLexemInfo.lexemStr)))

||

(!isLeftAssociative((\*lastLexemInfoInTable)->lexemStr) && (precedenceLevel((\*lastLexemInfoInTable)->lexemStr) < precedenceLevel(currLexemInfo.lexemStr)))

)) {

\*\*lastTempLexemInfoInTable = currLexemInfo; ++\* lastTempLexemInfoInTable;

--lexemInfoTransformationTempStackSize;

}

else {

break;

}

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*((\*lastLexemInfoInTable)++);

}

}

void makePrepare4LeftParenthesis(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable) {

if ((\*lastLexemInfoInTable)->lexemStr[0] == '(') {

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*((\*lastLexemInfoInTable)++);

}

}

void makePrepare4RightParenthesis(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable) {

if ((\*lastLexemInfoInTable)->lexemStr[0] == ')') {

bool findLeftParenthesis = false;

while (lexemInfoTransformationTempStackSize > 0) {

struct LexemInfo/\*&\*/ currLexemInfo = lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1];

if (currLexemInfo.lexemStr[0] == '(') {

findLeftParenthesis = true;

break;

}

else {

\*\*lastTempLexemInfoInTable = currLexemInfo; ++\* lastTempLexemInfoInTable;

lexemInfoTransformationTempStackSize--;

}

}

if (!findLeftParenthesis) {

printf("Warning: parentheses mismatched\n");

\*\*lastTempLexemInfoInTable = \*\*lastLexemInfoInTable; ++\* lastTempLexemInfoInTable;

}

else {

--lexemInfoTransformationTempStackSize;

}

++\* lastLexemInfoInTable;

}

}

unsigned int makePrepareEnder(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable) {

unsigned int addedLexemCount = (unsigned int)lexemInfoTransformationTempStackSize;

while (lexemInfoTransformationTempStackSize > 0) {

struct LexemInfo/\*&\*/ currLexemInfo = lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1];

if (currLexemInfo.lexemStr[0] == '(' || currLexemInfo.lexemStr[0] == ')') {

printf("Error: parentheses mismatched\n");

exit(0);

}

\*\*lastTempLexemInfoInTable = currLexemInfo, ++(\*lastTempLexemInfoInTable); // \*(\*lastTempLexemInfoInTable)++ = currLexemInfo;

--lexemInfoTransformationTempStackSize;

}

(\*lastTempLexemInfoInTable)->lexemStr[0] = '\0';

return addedLexemCount;

}

long long int getPrevNonParenthesesIndex(struct LexemInfo\* lexemInfoInTable, unsigned long long currIndex) {

if (!currIndex) {

return currIndex;

}

long long int index = currIndex - 1;

for (; index != ~0 && (

lexemInfoInTable[index].lexemStr[0] == '('

|| lexemInfoInTable[index].lexemStr[0] == ')'

);

--index);

return index;

}

long long int getEndOfNewPrevExpressioIndex(struct LexemInfo\* lexemInfoInTable, unsigned long long currIndex) {

if (!currIndex) { // || lexemInfoInTable[currIndex - 1].lexemStr[0] != '('

return currIndex;

}

long long int index = currIndex - 1;

for (; index != ~0 && lexemInfoInTable[index].lexemStr[0] == '(';

--index);

return index;

}

unsigned long long int getNextEndOfExpressionIndex(struct LexemInfo\* lexemInfoInTable, unsigned long long prevEndOfExpressionIndex) {

bool isPreviousExpressionComplete = false;

for (unsigned long long int index = prevEndOfExpressionIndex + 2; lexemInfoInTable[index].lexemStr[0] != '\0'; ++index) {

if (!strncmp(lexemInfoInTable[index].lexemStr, "(", MAX\_LEXEM\_SIZE) || !strncmp(lexemInfoInTable[index].lexemStr, ")", MAX\_LEXEM\_SIZE)) {

continue;

}

long long int prevNonParenthesesIndex = getPrevNonParenthesesIndex(lexemInfoInTable, index);

if (lexemInfoInTable[index].tokenType == IDENTIFIER\_LEXEME\_TYPE || lexemInfoInTable[index].tokenType == VALUE\_LEXEME\_TYPE) {

if (lexemInfoInTable[prevNonParenthesesIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE || lexemInfoInTable[prevNonParenthesesIndex].tokenType == VALUE\_LEXEME\_TYPE) {

return getEndOfNewPrevExpressioIndex(lexemInfoInTable, index);

}

}

else if (precedenceLevel(lexemInfoInTable[index].lexemStr) && isLeftAssociative(lexemInfoInTable[index].lexemStr)) {

if (precedenceLevel(lexemInfoInTable[prevNonParenthesesIndex].lexemStr)) {

return getEndOfNewPrevExpressioIndex(lexemInfoInTable, index);

}

}

else if (isSplittingOperator(lexemInfoInTable[index].lexemStr)) {

if (lexemInfoInTable[prevNonParenthesesIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE || lexemInfoInTable[prevNonParenthesesIndex].tokenType == VALUE\_LEXEME\_TYPE) {

return getEndOfNewPrevExpressioIndex(lexemInfoInTable, index);

}

}

else if (lexemInfoInTable[index].tokenType != IDENTIFIER\_LEXEME\_TYPE && lexemInfoInTable[index].tokenType != VALUE\_LEXEME\_TYPE && !precedenceLevel(lexemInfoInTable[index].lexemStr)) {

if (lexemInfoInTable[prevNonParenthesesIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE || lexemInfoInTable[prevNonParenthesesIndex].tokenType == VALUE\_LEXEME\_TYPE || precedenceLevel(lexemInfoInTable[prevNonParenthesesIndex].lexemStr)) {

return getEndOfNewPrevExpressioIndex(lexemInfoInTable, index);

}

}

}

return ~0;

}

void makePrepare(struct LexemInfo\* lexemInfoInTable, struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable) {

unsigned long long int nullStatementIndex = 0;

unsigned long long int passMakePrepareElementCount = getDataSectionLastLexemIndex(\*lastLexemInfoInTable, &grammar);

if (passMakePrepareElementCount++ == ~0) {

printf("Error: bad section!\r\n");

exit(0);

}

// \*lastLexemInfoInTable += lastDataSectionLexemIndex;

// while (lastDataSectionLexemIndex--) {

//

// }

//

// for (; false && (\*lastLexemInfoInTable)->lexemStr[0] != '\0'; \*(\*lastTempLexemInfoInTable)++ = \*(\*lastLexemInfoInTable)++) {

// if (passMakePrepareElementCount) {

// --passMakePrepareElementCount;

// ++lexemInfoInTable;

// continue;

// }

// else {

// break;

// }

// }

lexemInfoTransformationTempStackSize = 0;

for (; (\*lastLexemInfoInTable)->lexemStr[0] != '\0'; \*(\*lastTempLexemInfoInTable)++ = \*(\*lastLexemInfoInTable)++) {

if (passMakePrepareElementCount) {

--passMakePrepareElementCount;

++lexemInfoInTable;

continue;

}

for (struct LexemInfo\* lastLexemInfoInTable\_ = NULL; lastLexemInfoInTable\_ != \*lastLexemInfoInTable;) {

lastLexemInfoInTable\_ = \*lastLexemInfoInTable;

makePrepare4IdentifierOrValue(lastLexemInfoInTable, lastTempLexemInfoInTable);

if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable)

makePrepare4Operators(lastLexemInfoInTable, lastTempLexemInfoInTable);

if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable)

makePrepare4LeftParenthesis(lastLexemInfoInTable, lastTempLexemInfoInTable);

if (lastLexemInfoInTable\_ == \*lastLexemInfoInTable)

makePrepare4RightParenthesis(lastLexemInfoInTable, lastTempLexemInfoInTable);

if (lastLexemInfoInTable\_ != \*lastLexemInfoInTable

&& (!nullStatementIndex || (lexemInfoInTable + nullStatementIndex == lastLexemInfoInTable\_))) {

if (nullStatementIndex != ~0) {

if (nullStatementIndex) {

// printf("Added null statement after %lld(lexem index)\r\n", nullStatementIndex);

makePrepareEnder(lastLexemInfoInTable, lastTempLexemInfoInTable);

(void)createMultiToken(lastTempLexemInfoInTable, MULTI\_TOKEN\_NULL\_STATEMENT);

}

nullStatementIndex = getNextEndOfExpressionIndex(lexemInfoInTable, nullStatementIndex);

}

}

}

makePrepareEnder(lastLexemInfoInTable, lastTempLexemInfoInTable);

if ((!nullStatementIndex || (lexemInfoInTable + nullStatementIndex == \*lastLexemInfoInTable))) {

if (nullStatementIndex != ~0) {

if (nullStatementIndex) {

// printf("Added null statement after %lld(lexem index)\r\n", nullStatementIndex);

makePrepareEnder(lastLexemInfoInTable, lastTempLexemInfoInTable);

(void)createMultiToken(lastTempLexemInfoInTable, MULTI\_TOKEN\_NULL\_STATEMENT);

}

nullStatementIndex = getNextEndOfExpressionIndex(lexemInfoInTable, nullStatementIndex);

}

}

}

}  
  
  
repeat\_until.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: repeat\_until.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

#include "string.h"

unsigned char\* makeRepeatCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_REPEAT);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_REPEAT][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_REPEAT][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned long long int)currBytePtr;

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@REPEAT\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_REPEAT\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeUntileCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) { // Or Ender!

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_UNTIL);

if (multitokenSize

&& lexemInfoTransformationTempStackSize

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_REPEAT][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;%s\r\n", tokenStruct[MULTI\_TOKEN\_UNTIL][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //%s\r\n", tokenStruct[MULTI\_TOKEN\_UNTIL][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeNullStatementAfterUntilCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_NULL\_STATEMENT);

if (multitokenSize) {

if (lexemInfoTransformationTempStackSize < 2

|| strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_UNTIL][0], MAX\_LEXEM\_SIZE)

|| strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_REPEAT][0], MAX\_LEXEM\_SIZE)

) {

return currBytePtr;

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_cmp\_eax\_0[] = { 0x83, 0xF8, 0x00 };

const unsigned char code\_\_jnz\_offset[] = { 0x0F, 0x85, 0x00, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_eax\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jnz\_offset, 6);

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)((unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue - currBytePtr);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;after cond expresion (after \"%s\" after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_UNTIL][0], tokenStruct[MULTI\_TOKEN\_REPEAT][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " cmp eax, 0\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jnz LABEL@REPEAT\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //after cond expresion (after \"%s\" after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_UNTIL][0], tokenStruct[MULTI\_TOKEN\_REPEAT][0]);

//

currBytePtr += snprintf((char\*)currBytePtr, 8192, " if(opTemp != 0) goto LABEL\_\_REPEAT\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

}

lexemInfoTransformationTempStackSize -= 2;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
rlbind.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: rlbind.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeRightToLeftBindCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_RLBIND);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_mov\_ebx\_stackTopByECXMinus4[] = { 0x8B, 0x59, 0xFC };

const unsigned char code\_\_sub\_ecx\_8[] = { 0x83, 0xE9, 0x08 };

const unsigned char code\_\_add\_ebx\_edi[] = { 0x03, 0xDF };

const unsigned char code\_\_mov\_addrFromEBX\_eax[] = { 0x89, 0x03 };

const unsigned char code\_\_mov\_ecx\_edi[] = { 0x8B, 0xCF };

const unsigned char code\_\_add\_ecx\_512[] = { 0x81, 0xC1, 0x00, 0x02, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ebx\_stackTopByECXMinus4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_8, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ebx\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_addrFromEBX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_ecx\_edi, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_add\_ecx\_512, 6);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_RLBIND][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ebx, dword ptr[ecx - 4]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 8\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ebx, edi\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov dword ptr [ebx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov ecx, edi ; reset second stack\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " add ecx, 512 ; reset second stack\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_RLBIND][0]);

currBytePtr += sprintf((char\*)currBytePtr, " lastBindDataIndex = opStack[opStackIndex - 1];\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " data[lastBindDataIndex] = opTemp = opStack[opStackIndex], opStackIndex = 0;\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
semantix.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: semantix.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//#include "../../include/config.h"

#include "../../include/syntax/syntax.h"

#include "../../include/semantix/semantix.h"

#include "stdio.h"

#include "string.h"

//#include <iterator>

#include <regex>

//

//#define COLLISION\_II\_STATE 128

//#define COLLISION\_LL\_STATE 129

//#define COLLISION\_IL\_STATE 130

//#define COLLISION\_I\_STATE 132

//#define COLLISION\_L\_STATE 133

//

//#define NO\_IMPLEMENT\_CODE\_STATE 256

unsigned long long int getDataSectionLastLexemIndex(LexemInfo\* lexemInfoTable, Grammar\* grammar) {

int lexemIndex = 0;

const struct LexemInfo\* unexpectedLexemfailedTerminal = nullptr;

if (recursiveDescentParserRuleWithDebug("program\_\_\_\_part1", lexemIndex, lexemInfoTable, grammar, 0, &unexpectedLexemfailedTerminal)

&& lexemInfoTable[lexemIndex].lexemStr[0] != '\0') {

return lexemIndex;

}

printf("Error: No find data section end index!\r\n");

return ~0;

}

int checkingInternalCollisionInDeclarations(LexemInfo\* lexemInfoTable, Grammar\* grammar, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char \*\* errorMessagesPtrToLastBytePtr) {

// int returnState = SUCCESS\_STATE;

unsigned long long int lastDataSectionLexemIndex = 0;

if (~0 == (lastDataSectionLexemIndex = getDataSectionLastLexemIndex(lexemInfoTable, grammar))) { // TODO: ADD TO START CODE

\*errorMessagesPtrToLastBytePtr += sprintf(\*errorMessagesPtrToLastBytePtr, "Error get of data section last lexem index.\r\n");

return ~SUCCESS\_STATE;

}

for (unsigned int index = 0; identifierIdsTable[index][0] != '\0'; ++index) {

char isDeclaredIdentifier = 0;

char isDeclaredIdentifierCollision = 0;

unsigned int lexemIndex = 0;

for (lexemIndex = 0; lexemIndex <= lastDataSectionLexemIndex; ++lexemIndex) {

if (lexemesInfoTable[lexemIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE) {

if (!strncmp(identifierIdsTable[index], lexemesInfoTable[lexemIndex].lexemStr, MAX\_LEXEM\_SIZE)) {

if (isDeclaredIdentifier) {

isDeclaredIdentifierCollision = 1;

}

isDeclaredIdentifier = 1;

}

}

}

char isLabel = 0;

char isDeclaredLabel = 0;

char isDeclaredLabelCollision = 0;

for (unsigned int lexemIndex = 0; lexemesInfoTable[lexemIndex].lexemStr[0] != '\0'; ++lexemIndex) {

if (lexemesInfoTable[lexemIndex].tokenType != IDENTIFIER\_LEXEME\_TYPE || strncmp(identifierIdsTable[index], lexemesInfoTable[lexemIndex].lexemStr, MAX\_LEXEM\_SIZE)) {

continue;

}

if (!strncmp(lexemesInfoTable[lexemIndex + 1].lexemStr, tokenStruct[MULTI\_TOKEN\_COLON][0], MAX\_LEXEM\_SIZE)) {

if (isDeclaredLabel) {

isDeclaredLabelCollision = 1;

}

isLabel = 1;

isDeclaredLabel = 1;

}

if (lexemIndex && !strncmp(lexemesInfoTable[lexemIndex - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_GOTO][0], MAX\_LEXEM\_SIZE)) {

isLabel = 1;

}

}

// //tryToGetKeyWord(struct LexemInfo\* lexemInfoInTable);

// if (SUCCESS\_STATE != checkingCollisionInDeclarationsByKeyWords(identifierIdsTable[index])) {

// return COLLISION\_IK\_STATE;

// }

if (isDeclaredIdentifierCollision) {

printf("Collision(identifier/identifier): %s\r\n", identifierIdsTable[index]);

\*errorMessagesPtrToLastBytePtr += snprintf(\*errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + strlen("Collision(identifier/identifier): #\r\n"), "Collision(identifier/identifier): %s\r\n", identifierIdsTable[index]);

return COLLISION\_II\_STATE;

}

if (isDeclaredLabelCollision) {

printf("Collision(label/label): %s\r\n", identifierIdsTable[index]);

\*errorMessagesPtrToLastBytePtr += snprintf(\*errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + strlen("Collision(label/label): #\r\n"), "Collision(label/label): %s\r\n", identifierIdsTable[index]);

return COLLISION\_LL\_STATE;

}

if (isDeclaredIdentifier && isLabel) {

printf("Collision(identifier/label): %s\r\n", identifierIdsTable[index]);

\*errorMessagesPtrToLastBytePtr += snprintf(\*errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + strlen("Collision(identifier/label): #\r\n"), "Collision(identifier/label): %s\r\n", identifierIdsTable[index]);

return COLLISION\_IL\_STATE;

}

else if (!isDeclaredIdentifier && !isLabel && !isDeclaredLabel) {

printf("Undeclared identifier: %s\r\n", identifierIdsTable[index]);

\*errorMessagesPtrToLastBytePtr += snprintf(\*errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + strlen("Undeclared identifier: #\r\n"), "Undeclared identifier: %s\r\n", identifierIdsTable[index]);

return COLLISION\_I\_STATE;

}

else if (isLabel && !isDeclaredLabel) {

printf("Undeclared label: %s\r\n", identifierIdsTable[index]);

\*errorMessagesPtrToLastBytePtr += snprintf(\*errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + strlen("Undeclared label: #\r\n"), "Undeclared label: %s\r\n", identifierIdsTable[index]);

return COLLISION\_L\_STATE;

}

}

// if (returnState == SUCCESS\_STATE) {

printf("Declaration verification was successful!\r\n");

// }

//

return SUCCESS\_STATE;

}

int checkingVariableInitialization(LexemInfo\* lexemInfoTable, Grammar\* grammar, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\*\* errorMessagesPtrToLastBytePtr) {

int returnState = SUCCESS\_STATE;

unsigned long long int lastDataSectionLexemIndex = 0;

if (~0 == (lastDataSectionLexemIndex = getDataSectionLastLexemIndex(lexemInfoTable, grammar))) { // TODO: ADD TO START CODE

\*errorMessagesPtrToLastBytePtr += sprintf(\*errorMessagesPtrToLastBytePtr, "Error get of data section last lexem index.\r\n");

return ~SUCCESS\_STATE;

}

for (unsigned int index = 0; identifierIdsTable[index][0] != '\0'; ++index) {

for (unsigned int lexemIndex = lastDataSectionLexemIndex; lexemesInfoTable[lexemIndex].lexemStr[0] != '\0'; ++lexemIndex) {

if (lexemesInfoTable[lexemIndex].tokenType != IDENTIFIER\_LEXEME\_TYPE || strncmp(identifierIdsTable[index], lexemesInfoTable[lexemIndex].lexemStr, MAX\_LEXEM\_SIZE)) {

continue;

}

if (!strncmp(lexemesInfoTable[lexemIndex + 1].lexemStr, tokenStruct[MULTI\_TOKEN\_COLON][0], MAX\_LEXEM\_SIZE)) {

continue;

}

if (lexemIndex && !strncmp(lexemesInfoTable[lexemIndex - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_GOTO][0], MAX\_LEXEM\_SIZE)) {

continue;

}

int prevNonOpenParenthesesIndex = -1;

for (; !strncmp(lexemesInfoTable[lexemIndex + prevNonOpenParenthesesIndex].lexemStr, "(", MAX\_LEXEM\_SIZE); --prevNonOpenParenthesesIndex);

if (!strncmp(lexemesInfoTable[lexemIndex + 1].lexemStr, tokenStruct[MULTI\_TOKEN\_RLBIND][0], MAX\_LEXEM\_SIZE)

||

!strncmp(lexemesInfoTable[lexemIndex - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_LRBIND][0], MAX\_LEXEM\_SIZE)

||

//!strncmp(lexemesInfoTable[-1].lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)

//||

//!strncmp(lexemesInfoTable[-2].lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)

//||

!strncmp(lexemesInfoTable[lexemIndex + prevNonOpenParenthesesIndex].lexemStr, tokenStruct[MULTI\_TOKEN\_INPUT][0], MAX\_LEXEM\_SIZE)

){

break;

}

printf("Uninitialized: %s\r\n", identifierIdsTable[index]);

\*errorMessagesPtrToLastBytePtr += snprintf(\*errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + strlen("Uninitialized: #\r\n"), "Uninitialized: %s\r\n", identifierIdsTable[index]);

returnState = UNINITIALIZED\_I\_STATE;

break;

}

}

if (returnState == SUCCESS\_STATE) {

printf("Variable initialization checking was successful!\r\n");

}

return returnState;

}

int checkingCollisionInDeclarationsByKeyWords(char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\*\* errorMessagesPtrToLastBytePtr) {

int returnState = SUCCESS\_STATE;

char keywords\_re[] = KEYWORDS\_RE;

char keywords\_[sizeof(keywords\_re)] = { '\0' };

prepareKeyWordIdGetter(keywords\_, keywords\_re);

for (unsigned int index = 0; identifierIdsTable[index][0] != '\0'; ++index) {

if (std::regex\_match(std::string(identifierIdsTable[index]), std::regex(keywords\_re))) {

printf("Declaration matches keyword: %s\r\n", identifierIdsTable[index]);

\*errorMessagesPtrToLastBytePtr += snprintf(\*errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + strlen("Declaration matches keyword: #\r\n"), "Declaration matches keyword: %s\r\n", identifierIdsTable[index]);

returnState = COLLISION\_IK\_STATE;

}

}

printf("Declaration verification for keyword collision was successful!\r\n");

return SUCCESS\_STATE;

}

int semantixAnalyze(LexemInfo\* lexemInfoTable, Grammar\* grammar, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\* errorMessagesPtrToLastBytePtr){

int returnState = SUCCESS\_STATE;

if ( SUCCESS\_STATE != (returnState = checkingInternalCollisionInDeclarations(lexemesInfoTable, grammar, identifierIdsTable, &errorMessagesPtrToLastBytePtr))

|| SUCCESS\_STATE != (returnState = checkingVariableInitialization(lexemesInfoTable, grammar, identifierIdsTable, &errorMessagesPtrToLastBytePtr))

|| SUCCESS\_STATE != (returnState = checkingCollisionInDeclarationsByKeyWords(identifierIdsTable, &errorMessagesPtrToLastBytePtr))

) {

return returnState;

}

return SUCCESS\_STATE;

}  
  
semicolon.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: semicolon.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeSemicolonAfterNonContextCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_SEMICOLON);

if (multitokenSize

&&

!lexemInfoTransformationTempStackSize // !

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0]);

}

\* lastLexemInfoInTable += multitokenSize;

}

return currBytePtr;

}

unsigned char\* makeSemicolonIgnoreContextCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_SEMICOLON);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_SEMICOLON][0]);

}

\* lastLexemInfoInTable += multitokenSize;

}

return currBytePtr;

}  
  
  
sub.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: sub.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

unsigned char\* makeSubCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_SUB);

if (multitokenSize) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

const unsigned char code\_\_sub\_ecx\_4[] = { 0x83, 0xE9, 0x04 };

const unsigned char code\_\_sub\_stackTopByECX\_eax[] = { 0x29, 0x01 };

//const unsigned char code\_\_mov\_eax\_stackTopByECX[] = { 0x8B, 0x01 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_ecx\_4, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_sub\_stackTopByECX\_eax, 2);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_mov\_eax\_stackTopByECX, 2);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_SUB][0]);

//

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub ecx, 4\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " sub dword ptr[ecx], eax\r\n");

currBytePtr += sprintf((char\*)currBytePtr, " mov eax, dword ptr[ecx]\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_SUB][0]);

currBytePtr += sprintf((char\*)currBytePtr, " opTemp = opStack[opStackIndex - 1] -= opStack[opStackIndex--];\r\n");

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
  
syntax.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: syntax.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/config.h"

#include "../../include/syntax/syntax.h"

#include <iostream>

#include <fstream>

#include <iomanip>

#include <vector>

#include <map>

//#include <unordered\_map>

#include <string>

#include <set>

using namespace std;

Grammar grammar = {

CONFIGURABLE\_GRAMMAR

#if 0

{

{"labeled\_point", 2, {"ident", "tokenCOLON"}}, // !!!!!

{"goto\_label", 2, {"tokenGOTO","ident"}}, // !!!!!

{"program\_name", 1, {"ident\_terminal"}},

{"value\_type", 1, {T\_DATA\_TYPE\_0}},

{"other\_declaration\_ident", 2, {"tokenCOMMA", "ident"}},

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"other\_declaration\_ident","other\_declaration\_ident\_\_\_\_iteration\_after\_one", }},

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"tokenCOMMA", "ident"}},

{"value\_type\_\_ident", 2, {"value\_type", "ident"}},

{"declaration", 2, {"value\_type\_\_ident", "other\_declaration\_ident\_\_\_\_iteration\_after\_one"}},

{"declaration", 2, {"value\_type", "ident"}},

//

{"unary\_operator", 1, {T\_NOT\_0}},

{"unary\_operator", 1, {T\_SUB\_0}},

{"unary\_operator", 1, {T\_ADD\_0}},

{"binary\_operator", 1, {T\_AND\_0}},

{"binary\_operator", 1, {T\_OR\_0}},

{"binary\_operator", 1, {T\_EQUAL\_0}},

{"binary\_operator", 1, {T\_NOT\_EQUAL\_0}},

{"binary\_operator", 1, {T\_LESS\_OR\_EQUAL\_0}},

{"binary\_operator", 1, {T\_GREATER\_OR\_EQUAL\_0}},

{"binary\_operator", 1, {T\_ADD\_0}},

{"binary\_operator", 1, {T\_SUB\_0}},

{"binary\_operator", 1, {T\_MUL\_0}},

{"binary\_operator", 1, {T\_DIV\_0}},

{"binary\_operator", 1, {T\_MOD\_0}},

{"binary\_action", 2, {"binary\_operator","expression"}},

//

{"left\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},

{"left\_expression", 2, {"unary\_operator","expression"}},

{"left\_expression", 1, {"ident\_terminal"}},

{"left\_expression", 1, {"value\_terminal"}},

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action\_\_\_\_iteration\_after\_two"}},

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action"}},

{"expression", 2, {"left\_expression","binary\_action\_\_\_\_iteration\_after\_two"}},

{"expression", 2, {"left\_expression","binary\_action"}},

{"expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},

{"expression", 2, {"unary\_operator","expression"}},

{"expression", 1, {"ident\_terminal"}},

{"expression", 1, {"value\_terminal"}},

//

{"tokenGROUPEXPRESSIONBEGIN\_\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN","expression"}},

{"group\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},

//

{"bind\_right\_to\_left", 2, {"ident","rl\_expression"}},

{"bind\_left\_to\_right", 2, {"lr\_expression","ident"}},

//

{"body\_for\_true", 2, {"statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},

{"body\_for\_true", 2, {"statement\_in\_while\_body","tokenSEMICOLON"}},

{"body\_for\_true", 1, {T\_SEMICOLON\_0}},

{"tokenELSE\_\_statement\_in\_while\_body", 2, {"tokenELSE","statement\_in\_while\_body"}},

{"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenELSE","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body","tokenSEMICOLON"}},

{"body\_for\_false", 2, {"tokenELSE","tokenSEMICOLON"}},

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN", 2, {"tokenIF","tokenGROUPEXPRESSIONBEGIN"}},

{"expression\_\_tokenGROUPEXPRESSIONEND", 2, {"expression","tokenGROUPEXPRESSIONEND"}},

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN","expression\_\_tokenGROUPEXPRESSIONEND"}},

{"body\_for\_true\_\_body\_for\_false", 2, {"body\_for\_true","body\_for\_false"}},

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},

//

{"cycle\_counter", 1, {"ident\_terminal"}},

{"rl\_expression", 2, {"tokenRLBIND","expression"}},

{"lr\_expression", 2, {"expression","tokenLRBIND"}},

{"cycle\_counter\_init", 2, {"cycle\_counter","rl\_expression"}},

{"cycle\_counter\_init", 2, {"lr\_expression","cycle\_counter"}},

{"cycle\_counter\_last\_value", 1, {"value\_terminal"}},

{"cycle\_body", 2, {"tokenDO","statement\_\_\_\_iteration\_after\_two"}},

{"cycle\_body", 2, {"tokenDO","statement"}},

{"tokenFOR\_\_cycle\_counter\_init", 2, {"tokenFOR","cycle\_counter\_init"}},

{"tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenTO","cycle\_counter\_last\_value"}},

{"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenFOR\_\_cycle\_counter\_init","tokenTO\_\_cycle\_counter\_last\_value"}},

{"cycle\_body\_\_tokenSEMICOLON", 2, {"cycle\_body","tokenSEMICOLON"}},

{"forto\_cycle", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},

//

{"continue\_while", 2, {"tokenCONTINUE","tokenWHILE"}},

{"exit\_while", 2, {"tokenEXIT","tokenWHILE"}},

{"tokenWHILE\_\_expression", 2, {"tokenWHILE","expression"}},

{"tokenEND\_\_tokenWHILE", 2, {"tokenEND","tokenWHILE"}},

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body"}},

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE "}},

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},

{"while\_cycle", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},

//

{"tokenUNTIL\_\_expression", 2, {"tokenUNTIL","expression"}},

{"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two", 2, {"tokenREPEAT","statement\_\_\_\_iteration\_after\_two"}},

{"tokenREPEAT\_\_statement", 2, {"tokenREPEAT","statement"}},

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},

{"repeat\_until\_cycle", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},

//

{"input\_\_first\_part", 2, {"tokenGET","tokenGROUPEXPRESSIONBEGIN"}},

{"input\_\_second\_part", 2, {"ident","tokenGROUPEXPRESSIONEND"}},

{"input", 2, {"input\_\_first\_part","input\_\_second\_part"}},

//

{"output\_\_first\_part", 2, {"tokenPUT","tokenGROUPEXPRESSIONBEGIN"}},

{"output\_\_second\_part", 2, {"expression","tokenGROUPEXPRESSIONEND"}},

{"output", 2, {"output\_\_first\_part","output\_\_second\_part"}},

//

{"statement", 2, {"ident","rl\_expression"}},

{"statement", 2, {"lr\_expression","ident"}},

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},

{"statement", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},

{"statement", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},

{"statement", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},

{"statement", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},

{"statement", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},

{"statement", 2, {"ident","tokenCOLON"}},

{"statement", 2, {"tokenGOTO","ident"}},

{"statement", 2, {"input\_\_first\_part","input\_\_second\_part"}},

{"statement", 2, {"output\_\_first\_part","output\_\_second\_part"}},

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement\_\_\_\_iteration\_after\_two"}},

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement"}},

//

{ "statement\_in\_while\_body", 2, {"ident","rl\_expression"} },

{ "statement\_in\_while\_body", 2, {"lr\_expression","ident"} },

{ "statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"} },

{ "statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"} },

{ "statement\_in\_while\_body", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"} },

{ "statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"} },

{ "statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"} },

{ "statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"} },

{ "statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"} },

{ "statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"} },

{ "statement\_in\_while\_body", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"} },

{ "statement\_in\_while\_body", 2, {"ident","tokenCOLON"} },

{ "statement\_in\_while\_body", 2, {"tokenGOTO","ident"} },

{ "statement\_in\_while\_body", 2, {"input\_\_first\_part","input\_\_second\_part"} },

{ "statement\_in\_while\_body", 2, {"output\_\_first\_part","output\_\_second\_part"} },

{ "statement\_in\_while\_body", 2, {"tokenCONTINUE","tokenWHILE"} },

{ "statement\_in\_while\_body", 2, {"tokenEXIT","tokenWHILE"} },

{ "statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"} },

{ "statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body"} },

//

{"tokenNAME\_\_program\_name", 2, {"tokenNAME","program\_name"}},

{"tokenSEMICOLON\_\_tokenBODY", 2, {"tokenSEMICOLON","tokenBODY"}},

{"tokenDATA\_\_declaration", 2, {"tokenDATA","declaration"}},

{"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY", 2, {"tokenNAME\_\_program\_name","tokenSEMICOLON\_\_tokenBODY"}},

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA\_\_declaration"}},

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA"}},

{"statement\_\_tokenEND", 2, {"statement","tokenEND"}},

{"statement\_\_\_\_iteration\_after\_two\_\_tokenEND", 2, {"statement\_\_\_\_iteration\_after\_two","tokenEND"}},

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_\_\_iteration\_after\_two\_\_tokenEND"}},

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_tokenEND"}},

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","tokenEND"}},

{"program", 2, {"program\_\_\_\_part1","program\_\_\_\_part2"}},

//

{"tokenCOLON", 1, {T\_COLON\_0}},

{"tokenGOTO", 1, {T\_GOTO\_0}},

{"tokenINTEGER16", 1, {T\_DATA\_TYPE\_0}},

{"tokenCOMMA", 1, {T\_COMA\_0}},

{"tokenNOT", 1, {T\_NOT\_0}},

{"tokenAND", 1, {T\_AND\_0}},

{"tokenOR", 1, {T\_OR\_0}},

{"tokenEQUAL", 1, {T\_EQUAL\_0}},

{"tokenNOTEQUAL", 1, {T\_NOT\_EQUAL\_0}},

{"tokenLESSOREQUAL", 1, {T\_LESS\_OR\_EQUAL\_0}},

{"tokenGREATEROREQUAL", 1, {T\_GREATER\_OR\_EQUAL\_0}},

{"tokenPLUS", 1, {T\_ADD\_0}},

{"tokenMINUS", 1, {T\_SUB\_0}},

{"tokenMUL", 1, {T\_MUL\_0}},

{"tokenDIV", 1, {T\_DIV\_0}},

{"tokenMOD", 1, {T\_MOD\_0}},

{"tokenGROUPEXPRESSIONBEGIN", 1, {"("}},

{"tokenGROUPEXPRESSIONEND", 1, {")"}},

{"tokenRLBIND", 1, {T\_RLBIND\_0}},

{"tokenLRBIND", 1, {T\_LRBIND\_0}},

{"tokenELSE", 1, {T\_ELSE\_0}},

{"tokenIF", 1, {T\_IF\_0}},

{"tokenDO", 1, {T\_DO\_0}},

{"tokenFOR", 1, {T\_FOR\_0}},

{"tokenTO", 1, {T\_TO\_0}},

{"tokenWHILE", 1, {T\_WHILE\_0}},

{"tokenCONTINUE", 1, {T\_CONTINUE\_WHILE\_0}},

{"tokenEXIT", 1, {T\_EXIT\_WHILE\_0}},

{"tokenREPEAT", 1, {T\_REPEAT\_0}},

{"tokenUNTIL", 1, {T\_UNTIL\_0}},

{"tokenGET", 1, {T\_INPUT\_0}},

{"tokenPUT", 1, {T\_OUTPUT\_0}},

{"tokenNAME", 1, {T\_NAME\_0}},

{"tokenBODY", 1, {T\_BODY\_0}},

{"tokenDATA", 1, {T\_DATA\_0}},

{"tokenEND", 1, {T\_END\_0}},

{"tokenSEMICOLON", 1, {T\_SEMICOLON\_0}},

//

{ "value", 1, {"value\_terminal"} },

//

{ "ident", 1, {"ident\_terminal"} },

//

// { "label", 1, {"ident\_terminal"} },

//

{ "", 2, {"",""} }

},

176,

"program"

#endif

};

Grammar originalGrammar = {

ORIGINAL\_GRAMMAR

#if 0

{

{"labeled\_point", 2, {"ident", "tokenCOLON"}}, // !!!!!

{"goto\_label", 2, {"tokenGOTO","ident"}}, // !!!!!

{"program\_name", 1, {"ident\_terminal"}},

{"value\_type", 1, {"INTEGER16"}},

{"other\_declaration\_ident", 2, {"tokenCOMMA", "ident"}},

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"other\_declaration\_ident","other\_declaration\_ident\_\_\_\_iteration\_after\_one", }},

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"tokenCOMMA", "ident"}},

{"value\_type\_\_ident", 2, {"value\_type", "ident"}},

{"declaration", 2, {"value\_type\_\_ident", "other\_declaration\_ident\_\_\_\_iteration\_after\_one"}},

{"declaration", 2, {"value\_type", "ident"}},

//

{"unary\_operator", 1, {"NOT"}},

{"unary\_operator", 1, {"-"}},

{"unary\_operator", 1, {"+"}},

{"binary\_operator", 1, {"AND"}},

{"binary\_operator", 1, {"OR"}},

{"binary\_operator", 1, {"=="}},

{"binary\_operator", 1, {"!="}},

{"binary\_operator", 1, {"<="}},

{"binary\_operator", 1, {">="}},

{"binary\_operator", 1, {"+"}},

{"binary\_operator", 1, {"-"}},

{"binary\_operator", 1, {"\*"}},

{"binary\_operator", 1, {"DIV"}},

{"binary\_operator", 1, {"MOD"}},

{"binary\_action", 2, {"binary\_operator","expression"}},

//

{"left\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},

{"left\_expression", 2, {"unary\_operator","expression"}},

{"left\_expression", 1, {"ident\_terminal"}},

{"left\_expression", 1, {"value\_terminal"}},

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action\_\_\_\_iteration\_after\_two"}},

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action"}},

{"expression", 2, {"left\_expression","binary\_action\_\_\_\_iteration\_after\_two"}},

{"expression", 2, {"left\_expression","binary\_action"}},

{"expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},

{"expression", 2, {"unary\_operator","expression"}},

{"expression", 1, {"ident\_terminal"}},

{"expression", 1, {"value\_terminal"}},

//

{"tokenGROUPEXPRESSIONBEGIN\_\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN","expression"}},

{"group\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},

//

{"bind\_right\_to\_left", 2, {"ident","rl\_expression"}},

{"bind\_left\_to\_right", 2, {"lr\_expression","ident"}},

//

{"body\_for\_true", 2, {"statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},

{"body\_for\_true", 2, {"statement\_in\_while\_body","tokenSEMICOLON"}},

{"body\_for\_true", 1, {";"}},

{"tokenELSE\_\_statement\_in\_while\_body", 2, {"tokenELSE","statement\_in\_while\_body"}},

{"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenELSE","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body","tokenSEMICOLON"}},

{"body\_for\_false", 2, {"tokenELSE","tokenSEMICOLON"}},

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN", 2, {"tokenIF","tokenGROUPEXPRESSIONBEGIN"}},

{"expression\_\_tokenGROUPEXPRESSIONEND", 2, {"expression","tokenGROUPEXPRESSIONEND"}},

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN","expression\_\_tokenGROUPEXPRESSIONEND"}},

{"body\_for\_true\_\_body\_for\_false", 2, {"body\_for\_true","body\_for\_false"}},

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},

//

{"cycle\_counter", 1, {"ident\_terminal"}},

{"rl\_expression", 2, {"tokenRLBIND","expression"}},

{"lr\_expression", 2, {"expression","tokenLRBIND"}},

{"cycle\_counter\_init", 2, {"cycle\_counter","rl\_expression"}},

{"cycle\_counter\_init", 2, {"lr\_expression","cycle\_counter"}},

{"cycle\_counter\_last\_value", 1, {"value\_terminal"}},

{"cycle\_body", 2, {"tokenDO","statement\_\_\_\_iteration\_after\_two"}},

{"cycle\_body", 2, {"tokenDO","statement"}},

{"tokenFOR\_\_cycle\_counter\_init", 2, {"tokenFOR","cycle\_counter\_init"}},

{"tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenTO","cycle\_counter\_last\_value"}},

{"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenFOR\_\_cycle\_counter\_init","tokenTO\_\_cycle\_counter\_last\_value"}},

{"cycle\_body\_\_tokenSEMICOLON", 2, {"cycle\_body","tokenSEMICOLON"}},

{"forto\_cycle", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},

//

{"continue\_while", 2, {"tokenCONTINUE","tokenWHILE"}},

{"exit\_while", 2, {"tokenEXIT","tokenWHILE"}},

{"tokenWHILE\_\_expression", 2, {"tokenWHILE","expression"}},

{"tokenEND\_\_tokenWHILE", 2, {"tokenEND","tokenWHILE"}},

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body"}},

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE "}},

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},

{"while\_cycle", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},

//

{"tokenUNTIL\_\_expression", 2, {"tokenUNTIL","expression"}},

{"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two", 2, {"tokenREPEAT","statement\_\_\_\_iteration\_after\_two"}},

{"tokenREPEAT\_\_statement", 2, {"tokenREPEAT","statement"}},

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},

{"repeat\_until\_cycle", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},

//

{"input\_\_first\_part", 2, {"tokenGET","tokenGROUPEXPRESSIONBEGIN"}},

{"input\_\_second\_part", 2, {"ident","tokenGROUPEXPRESSIONEND"}},

{"input", 2, {"input\_\_first\_part","input\_\_second\_part"}},

//

{"output\_\_first\_part", 2, {"tokenPUT","tokenGROUPEXPRESSIONBEGIN"}},

{"output\_\_second\_part", 2, {"expression","tokenGROUPEXPRESSIONEND"}},

{"output", 2, {"output\_\_first\_part","output\_\_second\_part"}},

//

{"statement", 2, {"ident","rl\_expression"}},

{"statement", 2, {"lr\_expression","ident"}},

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},

{"statement", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},

{"statement", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},

{"statement", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},

{"statement", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},

{"statement", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},

{"statement", 2, {"ident","tokenCOLON"}},

{"statement", 2, {"tokenGOTO","ident"}},

{"statement", 2, {"input\_\_first\_part","input\_\_second\_part"}},

{"statement", 2, {"output\_\_first\_part","output\_\_second\_part"}},

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement\_\_\_\_iteration\_after\_two"}},

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement"}},

//

{"statement\_in\_while\_body", 2, {"ident","rl\_expression"} },

{"statement\_in\_while\_body", 2, {"lr\_expression","ident"} },

{"statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"} },

{"statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"} },

{"statement\_in\_while\_body", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"} },

{"statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"} },

{"statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"} },

{"statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"} },

{"statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"} },

{"statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"} },

{"statement\_in\_while\_body", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"} },

{"statement\_in\_while\_body", 2, {"ident","tokenCOLON"} },

{"statement\_in\_while\_body", 2, {"tokenGOTO","ident"} },

{"statement\_in\_while\_body", 2, {"input\_\_first\_part","input\_\_second\_part"} },

{"statement\_in\_while\_body", 2, {"output\_\_first\_part","output\_\_second\_part"} },

{"statement\_in\_while\_body", 2, {"tokenCONTINUE","tokenWHILE"} },

{"statement\_in\_while\_body", 2, {"tokenEXIT","tokenWHILE"} },

{"statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"} },

{"statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body"} },

//

{"tokenNAME\_\_program\_name", 2, {"tokenNAME","program\_name"}},

{"tokenSEMICOLON\_\_tokenBODY", 2, {"tokenSEMICOLON","tokenBODY"}},

{"tokenDATA\_\_declaration", 2, {"tokenDATA","declaration"}},

{"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY", 2, {"tokenNAME\_\_program\_name","tokenSEMICOLON\_\_tokenBODY"}},

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA\_\_declaration"}},

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA"}},

{"statement\_\_tokenEND", 2, {"statement","tokenEND"}},

{"statement\_\_\_\_iteration\_after\_two\_\_tokenEND", 2, {"statement\_\_\_\_iteration\_after\_two","tokenEND"}},

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_\_\_iteration\_after\_two\_\_tokenEND"}},

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_tokenEND"}},

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","tokenEND"}},

{"program", 2, {"program\_\_\_\_part1","program\_\_\_\_part2"}},

//

{"tokenCOLON", 1, {":"}},

{"tokenGOTO", 1, {"GOTO"}},

{"tokenINTEGER16", 1, {"INTEGER16"}},

{"tokenCOMMA", 1, {","}},

{"tokenNOT", 1, {"NOT"}},

{"tokenAND", 1, {"AND"}},

{"tokenOR", 1, {"OR"}},

{"tokenEQUAL", 1, {"=="}},

{"tokenNOTEQUAL", 1, {"!="}},

{"tokenLESSOREQUAL", 1, {"<="}},

{"tokenGREATEROREQUAL", 1, {">="}},

{"tokenPLUS", 1, {"+"}},

{"tokenMINUS", 1, {"-"}},

{"tokenMUL", 1, {"\*"}},

{"tokenDIV", 1, {"DIV"}},

{"tokenMOD", 1, {"MOD"}},

{"tokenGROUPEXPRESSIONBEGIN", 1, {"("}},

{"tokenGROUPEXPRESSIONEND", 1, {")"}},

{"tokenRLBIND", 1, {"<<"}},

{"tokenLRBIND", 1, {">>"}},

{"tokenELSE", 1, {"ELSE"}},

{"tokenIF", 1, {"IF"}},

{"tokenDO", 1, {"DO"}},

{"tokenFOR", 1, {"FOR"}},

{"tokenTO", 1, {"TO"}},

{"tokenWHILE", 1, {"WHILE"}},

{"tokenCONTINUE", 1, {"CONTINUE"}},

{"tokenEXIT", 1, {"EXIT"}},

{"tokenREPEAT", 1, {"REPEAT"}},

{"tokenUNTIL", 1, {"UNTIL"}},

{"tokenGET", 1, {"GET"}},

{"tokenPUT", 1, {"PUT"}},

{"tokenNAME", 1, {"NAME"}},

{"tokenBODY", 1, {"BODY"}},

{"tokenDATA", 1, {"DATA"}},

{"tokenEND", 1, {"END"}},

{"tokenSEMICOLON", 1, {";"}},

//

{ "value", 1, {"value\_terminal"} },

//

{ "ident", 1, {"ident\_terminal"} },

//

// { "label", 1, {"ident\_terminal"} },

//

{ "", 2, {"",""} }

},

176,

"program"

#endif

};

#define DEBUG\_STATES

#define MAX\_LEXEMS 256

//#define MAX\_RULES 128

#define MAX\_SYMBOLS 64

typedef struct {

char symbols[MAX\_SYMBOLS][MAX\_TOKEN\_SIZE];

int count;

} SymbolSet;

typedef SymbolSet ParseInfoTable[MAX\_LEXEMS][MAX\_LEXEMS];

bool insertIntoSymbolSet(SymbolSet\* set, const char\* symbol) {

for (int i = 0; i < set->count; ++i) {

if (strcmp(set->symbols[i], symbol) == 0) {

// symbol already exists

return false;

}

}

strncpy(set->symbols[set->count], symbol, MAX\_TOKEN\_SIZE);

set->symbols[set->count][MAX\_TOKEN\_SIZE - 1] = '\0';

++set->count;

return true;

}

bool containsSymbolSet(const SymbolSet\* set, const char\* symbol) {

for (int i = 0; i < set->count; ++i) {

if (strcmp(set->symbols[i], symbol) == 0) {

return true;

}

}

return false;

}

// initialize with empty SymbolSets

ParseInfoTable parseInfoTable = { {{0}} };

struct ASTNode {

std::string value;

bool isTerminal;

std::vector<ASTNode\*> children;

ASTNode(const std::string& val, bool isTerminal) : isTerminal(isTerminal), value(val) {}

~ASTNode() {

for (ASTNode\* child : children) {

delete child;

}

}

};

ASTNode\* buildASTByCPPMap(const std::map<int, std::map<int, std::set<std::string>>>& parseInfoTable,

Grammar\* grammar,

int start,

int end,

const std::string& symbol) {

if (start > end) return nullptr;

ASTNode\* node = new ASTNode(symbol, false);

for (const Rule& rule : grammar->rules) {

if (rule.lhs != symbol) continue;

if (rule.rhs\_count == 1) {

//if (parseInfoTable.at(start).at(end).count(rule.rhs[0])) {

node->children.push\_back(new ASTNode(rule.rhs[0], true));

return node;

//}

}

else if (rule.rhs\_count == 2) {

for (int split = start; split < end; ++split) {

if (parseInfoTable.at(start).at(split).count(rule.rhs[0]) &&

parseInfoTable.at(split + 1).at(end).count(rule.rhs[1])) {

node->children.push\_back(buildASTByCPPMap(parseInfoTable, grammar, start, split, rule.rhs[0]));

node->children.push\_back(buildASTByCPPMap(parseInfoTable, grammar, split + 1, end, rule.rhs[1]));

return node;

}

}

}

}

return nullptr;

}

ASTNode\* buildAST(//const std::map<int, std::map<int, std::set<std::string>>>& parseInfoTable,

ParseInfoTable& parseInfoTable,

Grammar\* grammar,

int start,

int end,

const std::string& symbol) {

if (start > end) return nullptr;

ASTNode\* node = new ASTNode(symbol, false);

for (const Rule& rule : grammar->rules) {

if (rule.lhs != symbol) continue;

if (rule.rhs\_count == 1) {

//if (parseInfoTable.at(start).at(end).count(rule.rhs[0])) {

node->children.push\_back(new ASTNode(rule.rhs[0], true));

return node;

//}

}

else if (rule.rhs\_count == 2) {

for (int split = start; split < end; ++split) {

if (containsSymbolSet(&parseInfoTable[start][split], rule.rhs[0]) &&

containsSymbolSet(&parseInfoTable[split + 1][end], rule.rhs[1])) {

node->children.push\_back(buildAST(parseInfoTable, grammar, start, split, rule.rhs[0]));

node->children.push\_back(buildAST(parseInfoTable, grammar, split + 1, end, rule.rhs[1]));

return node;

}

}

}

}

return nullptr;

}

void printAST(struct LexemInfo\* lexemInfoTable, const ASTNode\* node, int depth = 0) {

static int lexemInfoTableIndexForPrintAST = 0; // ATTENTION: multithreading is not supported for this!

if (!node) {

return;

}

if (!depth) {

lexemInfoTableIndexForPrintAST = 0;

}

for (unsigned int depthIndex = 0; depthIndex <= depth; ++depthIndex) {

std::cout << " " << "|";

}

std::cout << "--";

if (node->isTerminal) {

std::cout << "\"" << lexemInfoTable[lexemInfoTableIndexForPrintAST++].lexemStr << "\"";

}

else {

std::cout << node->value;

}

std::cout << "\n";

for (const ASTNode\* child : node->children) {

printAST(lexemInfoTable, child, depth + 1);

}

}

void printASTToFile(struct LexemInfo\* lexemInfoTable, const ASTNode\* node, std::ofstream& outFile, int depth = 0) {

static int lexemInfoTableIndexForPrintAST = 0; // ATTENTION: multithreading is not supported for this!

if (!node) {

return;

}

if (!depth) {

lexemInfoTableIndexForPrintAST = 0;

}

for (unsigned int depthIndex = 0; depthIndex <= depth; ++depthIndex) {

outFile << " |";

}

outFile << "--";

if (node->isTerminal) {

outFile << "\"" << lexemInfoTable[lexemInfoTableIndexForPrintAST++].lexemStr << "\"";

}

else {

outFile << node->value;

}

outFile << "\n";

for (const ASTNode\* child : node->children) {

printASTToFile(lexemInfoTable, child, outFile, depth + 1);

}

}

void printAST\_\_OLD\_123(struct LexemInfo\* lexemInfoTable, const ASTNode\* node, int depth = 0) {

static int lexemInfoTableIndexForPrintAST = 0; // ATTENTION: multithreading is not supported for this!

if (!node) {

return;

}

if (!depth) {

lexemInfoTableIndexForPrintAST = 0;

}

for (unsigned int depthIndex = 0; depthIndex <= depth; ++depthIndex) {

std::cout << " " << "|";

}

std::cout << "--";

if (node->isTerminal) {

std::cout << "\"" << lexemInfoTable[lexemInfoTableIndexForPrintAST++].lexemStr << "\"";

}

else {

std::cout << node->value;

}

std::cout << "\n";

for (const ASTNode\* child : node->children) {

printAST(lexemInfoTable, child, depth + 1);

}

}

void displayParseInfoTable(const map<int, map<int, set<string>>>& parseInfoTable) {

constexpr int CELL\_WIDTH = 128;

cout << left << setw(CELL\_WIDTH) << "[i\\j]";

for (const auto& outerEntry : parseInfoTable) {

cout << setw(CELL\_WIDTH) << outerEntry.first;

}

cout << endl;

for (const auto& outerEntry : parseInfoTable) {

int i = outerEntry.first;

cout << setw(CELL\_WIDTH) << i;

for (const auto& innerEntry : parseInfoTable) {

int j = innerEntry.first;

if (parseInfoTable.at(i).find(j) != parseInfoTable.at(i).end()) {

const set<string>& rules = parseInfoTable.at(i).at(j);

string cellContent;

for (const string& rule : rules) {

cellContent += rule + ", ";

}

if (!cellContent.empty()) {

cellContent.pop\_back();

cellContent.pop\_back();

}

cout << setw(CELL\_WIDTH) << cellContent;

}

else {

cout << setw(CELL\_WIDTH) << "-";

}

}

cout << endl;

}

}

void saveParseInfoTableToFile(const map<int, map<int, set<string>>>& parseInfoTable, const string& filename) {

constexpr int CELL\_WIDTH = 128;

ofstream file(filename);

if (!file.is\_open()) {

cerr << "Error: Unable to open file " << filename << endl;

return;

}

file << left << setw(CELL\_WIDTH) << "[i\\j]";

for (const auto& outerEntry : parseInfoTable) {

file << setw(CELL\_WIDTH) << outerEntry.first;

}

file << endl;

for (const auto& outerEntry : parseInfoTable) {

int i = outerEntry.first;

file << setw(CELL\_WIDTH) << i;

for (const auto& innerEntry : parseInfoTable) {

int j = innerEntry.first;

if (parseInfoTable.at(i).find(j) != parseInfoTable.at(i).end()) {

const set<string>& rules = parseInfoTable.at(i).at(j);

string cellContent;

for (const string& rule : rules) {

cellContent += rule + ", ";

}

if (!cellContent.empty()) {

cellContent.pop\_back();

cellContent.pop\_back();

}

file << setw(CELL\_WIDTH) << cellContent;

}

else {

file << setw(CELL\_WIDTH) << "-";

}

}

file << endl;

}

file.close();

}

bool cykAlgorithmImplementation(struct LexemInfo\* lexemInfoTable, Grammar\* grammar, char \* astFileName) {

if (lexemInfoTable == NULL || grammar == NULL) {

return false;

}

#if defined(\_DEBUG)

printf("ATTENTION: for better performance, use Release mode!\r\n");

#endif

#ifndef DEBUG\_STATES

cout << "cykParse in progress.....[please wait]";

#else

cout << "cykParse in progress.....[please wait]: ";

#endif

// ParseInfoTable parseInfoTable = { {{0}} }; // Initialize with empty SymbolSets

int lexemIndex = 0;

for (--lexemIndex; lexemInfoTable[++lexemIndex].lexemStr[0];) {

#ifdef DEBUG\_STATES

printf("\rcykParse in progress.....[please wait]: %02d %16s", lexemIndex, lexemInfoTable[lexemIndex].lexemStr);

#endif

// Iterate over the rules

for (int xIndex = 0; xIndex < grammar->rule\_count; ++xIndex) {

Rule& rule = grammar->rules[xIndex];

// If a terminal is found

if (rule.rhs\_count == 1 && (

lexemInfoTable[lexemIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "ident\_terminal")

|| lexemInfoTable[lexemIndex].tokenType == VALUE\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "value\_terminal")

|| !strncmp(rule.rhs[0], lexemInfoTable[lexemIndex].lexemStr, MAX\_LEXEM\_SIZE)

)) {

insertIntoSymbolSet(&parseInfoTable[lexemIndex][lexemIndex], rule.lhs);

}

}

for (int iIndex = lexemIndex; iIndex >= 0; --iIndex) {

for (int kIndex = iIndex; kIndex <= lexemIndex; ++kIndex) {

for (int xIndex = 0; xIndex < grammar->rule\_count; ++xIndex) {

Rule& rule = grammar->rules[xIndex];

if (rule.rhs\_count == 2

&& containsSymbolSet(&parseInfoTable[iIndex][kIndex], rule.rhs[0])

&& containsSymbolSet(&parseInfoTable[kIndex + 1][lexemIndex], rule.rhs[1])

) {

insertIntoSymbolSet(&parseInfoTable[iIndex][lexemIndex], rule.lhs);

}

}

}

}

}

cout << "\r" << "cykParse complete........[ ok ]\n";

if (!containsSymbolSet(&parseInfoTable[0][lexemIndex - 1], grammar->start\_symbol)) {

return false;

}

ASTNode\* astRoot = buildAST(parseInfoTable, grammar, 0, lexemIndex - 1, grammar->start\_symbol);

if (astRoot) {

std::cout << "Abstract Syntax Tree:\n";

printAST(lexemInfoTable, astRoot);

if (astFileName && astFileName[0] != '\0') {

std::ofstream astOFStream(astFileName, std::ofstream::out);

printASTToFile(lexemInfoTable, astRoot, astOFStream);

astOFStream.close();

printf("File \"%s\" saved.\n", astFileName);

}

delete astRoot;

}

else {

std::cout << "Failed to build AST.\n";

}

return true;

}

#define MAX\_STACK\_DEPTH 256

bool recursiveDescentParserRuleWithDebug(const char\* ruleName, int& lexemIndex, LexemInfo\* lexemInfoTable, Grammar\* grammar, int depth, const struct LexemInfo\*\* unexpectedLexemfailedTerminal) {

if (depth > MAX\_STACK\_DEPTH) {

printf("Error: Maximum recursion depth reached.\n");

return false;

}

char isError = false;

for (int i = 0; i < grammar->rule\_count; ++i) {

Rule& rule = grammar->rules[i];

if (strcmp(rule.lhs, ruleName) != 0) continue;

int savedIndex = lexemIndex;

if (rule.rhs\_count == 1) {

if (

lexemInfoTable[lexemIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "ident\_terminal")

|| lexemInfoTable[lexemIndex].tokenType == VALUE\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "value\_terminal")

|| !strncmp(rule.rhs[0], lexemInfoTable[lexemIndex].lexemStr, MAX\_LEXEM\_SIZE)

) {

++lexemIndex;

return true;

}

else {

\*unexpectedLexemfailedTerminal = lexemInfoTable + lexemIndex;

if (0)printf("<< \"%s\" >>\n", rule.rhs[0]);

}

}

else if (rule.rhs\_count == 2) {

if (recursiveDescentParserRuleWithDebug(rule.rhs[0], lexemIndex, lexemInfoTable, grammar, depth + 1, unexpectedLexemfailedTerminal) &&

recursiveDescentParserRuleWithDebug(rule.rhs[1], lexemIndex, lexemInfoTable, grammar, depth + 1, unexpectedLexemfailedTerminal)) {

return true;

}

}

lexemIndex = savedIndex;

}

return false;

}

const LexemInfo\* recursiveDescentParserWithDebug\_(const char\* ruleName, int& lexemIndex, LexemInfo\* lexemInfoTable, Grammar\* grammar, int depth, const struct LexemInfo\* unexpectedUnknownLexemfailedTerminal) {

if (depth > MAX\_STACK\_DEPTH) {

printf("Error: Maximum recursion depth reached.\n");

return unexpectedUnknownLexemfailedTerminal;

}

char isError = false;

const LexemInfo\* currUnexpectedLexemfailedTerminalPtr = nullptr, \* returnUnexpectedLexemfailedTerminalPtr = nullptr;

for (int i = 0; i < grammar->rule\_count; ++i) {

Rule& rule = grammar->rules[i];

if (strcmp(rule.lhs, ruleName) != 0) continue;

int savedIndex = lexemIndex;

if (rule.rhs\_count == 1) {

if (

lexemInfoTable[lexemIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "ident\_terminal")

|| lexemInfoTable[lexemIndex].tokenType == VALUE\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "value\_terminal")

|| !strncmp(rule.rhs[0], lexemInfoTable[lexemIndex].lexemStr, MAX\_LEXEM\_SIZE)

) {

++lexemIndex;

return nullptr;

}

else {

currUnexpectedLexemfailedTerminalPtr = lexemInfoTable + lexemIndex;

}

}

else if (rule.rhs\_count == 2) {

if (nullptr == (returnUnexpectedLexemfailedTerminalPtr = recursiveDescentParserWithDebug\_(rule.rhs[0], lexemIndex, lexemInfoTable, grammar, depth + 1, unexpectedUnknownLexemfailedTerminal))

&& nullptr == (returnUnexpectedLexemfailedTerminalPtr = recursiveDescentParserWithDebug\_(rule.rhs[1], lexemIndex, lexemInfoTable, grammar, depth + 1, unexpectedUnknownLexemfailedTerminal))) {

return nullptr;

}

}

lexemIndex = savedIndex;

}

if (returnUnexpectedLexemfailedTerminalPtr != nullptr && returnUnexpectedLexemfailedTerminalPtr != unexpectedUnknownLexemfailedTerminal

&&( returnUnexpectedLexemfailedTerminalPtr->tokenType == IDENTIFIER\_LEXEME\_TYPE

|| returnUnexpectedLexemfailedTerminalPtr->tokenType == VALUE\_LEXEME\_TYPE

|| returnUnexpectedLexemfailedTerminalPtr->tokenType == KEYWORD\_LEXEME\_TYPE

)) {

return returnUnexpectedLexemfailedTerminalPtr;

}

if (currUnexpectedLexemfailedTerminalPtr != nullptr) {

return currUnexpectedLexemfailedTerminalPtr;

}

if(returnUnexpectedLexemfailedTerminalPtr != nullptr){

return returnUnexpectedLexemfailedTerminalPtr;

}

return unexpectedUnknownLexemfailedTerminal;

}

//

int syntaxAnalyze(LexemInfo\* lexemInfoTable, Grammar\* grammar, char syntaxlAnalyzeMode, char\* astFileName, char\* errorMessagesPtrToLastBytePtr) {

bool cykAlgorithmImplementationReturnValue = false;

if (syntaxlAnalyzeMode == SYNTAX\_ANALYZE\_BY\_CYK\_ALGORITHM) {

cykAlgorithmImplementationReturnValue = cykAlgorithmImplementation(lexemesInfoTable, grammar, astFileName);

//printf("cykAlgorithmImplementation return \"%s\".\r\n", cykAlgorithmImplementationReturnValue ? "true" : "false");

if (cykAlgorithmImplementationReturnValue) {

return SUCCESS\_STATE;

}

else {

writeBytesToFile(astFileName, (unsigned char\*)"Error of AST build", strlen("Error of AST build"));

}

}

else if (astFileName && astFileName[0] != '\0') {

writeBytesToFile(astFileName, (unsigned char\*)"AST build no support.", strlen("AST build no support."));

}

if (cykAlgorithmImplementationReturnValue == false || syntaxlAnalyzeMode == SYNTAX\_ANALYZE\_BY\_RECURSIVE\_DESCENT) {

int lexemIndex = 0;

const struct LexemInfo\* unexpectedLexemfailedTerminal = nullptr;

if (recursiveDescentParserRuleWithDebug(grammar->start\_symbol, lexemIndex, lexemInfoTable, grammar, 0, &unexpectedLexemfailedTerminal)) {

if (lexemInfoTable[lexemIndex].lexemStr[0] == '\0') {

printf("Parse successful.\n");

printf("%d.\n", lexemIndex);

return SUCCESS\_STATE;

}

else {

printf("Parse failed: Extra tokens remain.\r\n");

errorMessagesPtrToLastBytePtr += sprintf(errorMessagesPtrToLastBytePtr, "Parse failed: Extra tokens remain.\r\n");

return ~SUCCESS\_STATE;

}

}

else {

if (unexpectedLexemfailedTerminal) {

printf("Parse failed.\r\n");

printf(" (The predicted terminal does not match the expected one.\r\n Possible unexpected terminal \"%s\" on line %lld at position %lld\r\n ..., but this is not certain.)\r\n", unexpectedLexemfailedTerminal->lexemStr, unexpectedLexemfailedTerminal->row, unexpectedLexemfailedTerminal->col);

errorMessagesPtrToLastBytePtr += sprintf(errorMessagesPtrToLastBytePtr, "Parse failed.\r\n");

errorMessagesPtrToLastBytePtr += snprintf(errorMessagesPtrToLastBytePtr, MAX\_LEXEM\_SIZE + 128 + strlen(" (The predicted terminal does not match the expected one.\r\n Possible unexpected terminal \"#\" on line # at position #\r\n ..., but this is not certain.)\r\n"), " (The predicted terminal does not match the expected one.\r\n Possible unexpected terminal \"%s\" on line %lld at position %lld\r\n ..., but this is not certain.)\r\n", unexpectedLexemfailedTerminal->lexemStr, unexpectedLexemfailedTerminal->row, unexpectedLexemfailedTerminal->col);

}

else {

printf("Parse failed: unexpected terminal.\r\n");

errorMessagesPtrToLastBytePtr += sprintf(errorMessagesPtrToLastBytePtr, "Parse failed: unexpected terminal.\r\n");

}

return ~SUCCESS\_STATE;

}

}

return ~SUCCESS\_STATE;

}

bool syntaxlAnalyze\_(LexemInfo\* lexemInfoTable, Grammar\* grammar, char syntaxlAnalyzeMode, char\* astFileName, char\* errorMessagesPtrToLastBytePtr) {

bool cykAlgorithmImplementationReturnValue = false;

if (syntaxlAnalyzeMode == SYNTAX\_ANALYZE\_BY\_CYK\_ALGORITHM) {

bool cykAlgorithmImplementationReturnValue = cykAlgorithmImplementation(lexemesInfoTable, grammar, astFileName);

printf("cykAlgorithmImplementation return \"%s\".\r\n", cykAlgorithmImplementationReturnValue ? "true" : "false");

if(!cykAlgorithmImplementationReturnValue) {

writeBytesToFile(astFileName, (unsigned char\*)"Error of AST build", strlen("Error of AST build"));

}

}

else if(astFileName && astFileName[0] != '\0') {

writeBytesToFile(astFileName, (unsigned char\*)"AST build no support.", strlen("AST build no support."));

}

if (cykAlgorithmImplementationReturnValue && syntaxlAnalyzeMode == SYNTAX\_ANALYZE\_BY\_RECURSIVE\_DESCENT) {

int lexemIndex = 0;

const struct LexemInfo unexpectedUnknownLexemfailedTerminal("unknown", 0, 0, 0, ~0, ~0); //

const struct LexemInfo\* returnUnexpectedLexemfailedTerminal = nullptr;

if (nullptr == (returnUnexpectedLexemfailedTerminal = recursiveDescentParserWithDebug\_(grammar->start\_symbol, lexemIndex, lexemInfoTable, grammar, 0, &unexpectedUnknownLexemfailedTerminal))) {

if (lexemInfoTable[lexemIndex].lexemStr[0] == '\0') {

printf("Parse successful.\n");

printf("%d.\n", lexemIndex);

return true;

}

else {

printf("Parse failed: Extra tokens remain.\n");

return false;

}

}

else {

if (returnUnexpectedLexemfailedTerminal->lexemStr[1]) {

printf("Parse failed.\r\n");

printf(" (The predicted terminal does not match the expected one.\r\n Possible unexpected terminal \"%s\" on line %lld at position %lld\r\n ..., but this is not certain.)\r\n", returnUnexpectedLexemfailedTerminal->lexemStr, returnUnexpectedLexemfailedTerminal->row, returnUnexpectedLexemfailedTerminal->col);

}

else {

printf("Parse failed: unexpected terminal.\r\n");

}

return false;

}

return false;

}

return false;

}

// OLD //

bool cykAlgorithmImplementationByCPPMap(struct LexemInfo\* lexemInfoTable, Grammar\* grammar) {

if (lexemInfoTable == NULL || grammar == NULL) {

return false;

}

#if defined(\_DEBUG)

printf("ATTENTION: for better performance, use Release mode!\r\n");

#endif

#ifndef DEBUG\_STATES

cout << "cykParse in progress.....[please wait]";

#else

cout << "cykParse in progress.....[please wait]: ";

#endif

map<int, map<int, set<string>>> parseInfoTable;

int lexemIndex = 0;

for (--lexemIndex; lexemInfoTable[++lexemIndex].lexemStr[0];) {

#ifdef DEBUG\_STATES

printf("\rcykParse in progress.....[please wait]: %02d %16s", lexemIndex, lexemInfoTable[lexemIndex].lexemStr);

#endif

// Iterate over the rules

for (int xIndex = 0; xIndex < grammar->rule\_count; ++xIndex) {

string&& lhs = grammar->rules[xIndex].lhs;

Rule& rule = grammar->rules[xIndex];

// If a terminal is found

if (rule.rhs\_count == 1 && (

lexemInfoTable[lexemIndex].tokenType == IDENTIFIER\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "ident\_terminal")

|| lexemInfoTable[lexemIndex].tokenType == VALUE\_LEXEME\_TYPE && !strcmp(rule.rhs[0], "value\_terminal")

|| !strncmp(rule.rhs[0], lexemInfoTable[lexemIndex].lexemStr, MAX\_LEXEM\_SIZE)

)) {

parseInfoTable[lexemIndex][lexemIndex].insert(lhs);

}

}

for (int iIndex = lexemIndex; iIndex >= 0; --iIndex) {

for (int kIndex = iIndex; kIndex <= lexemIndex; ++kIndex) {

for (int xIndex = 0; xIndex < grammar->rule\_count; ++xIndex) {

string&& lhs = grammar->rules[xIndex].lhs;

Rule& rule = grammar->rules[xIndex];

if (rule.rhs\_count == 2

&& parseInfoTable[iIndex][kIndex].find(rule.rhs[0]) != parseInfoTable[iIndex][kIndex].end()

&& parseInfoTable[kIndex + 1][lexemIndex].find(rule.rhs[1]) != parseInfoTable[kIndex + 1][lexemIndex].end()

) {

parseInfoTable[iIndex][lexemIndex].insert(lhs);

}

}

}

}

}

cout << "\r" << "cykParse complete........[ ok ]\n";

if (parseInfoTable[0][lexemIndex - 1].find(grammar->start\_symbol) == parseInfoTable[0][lexemIndex - 1].end()) {

return false;

}

// parseByRecursiveDescent\_(lexemInfoTable, grammar);

// displayParseInfoTable(parseInfoTable);

// saveParseInfoTableToFile(parseInfoTable, "parseInfoTable.txt");

ASTNode\* astRoot = buildASTByCPPMap(parseInfoTable, grammar, 0, lexemIndex - 1, grammar->start\_symbol);

if (astRoot) {

std::cout << "Abstract Syntax Tree:\n";

printAST(lexemInfoTable, astRoot);

delete astRoot; // Не забуваємо звільняти пам'ять

}

else {

std::cout << "Failed to build AST.\n";

}

//return parseInfoTable[0][lexemIndex - 1].find(grammar->start\_symbol) != parseInfoTable[0][lexemIndex - 1].end(); // return !!parseInfoTable[0][lexemIndex - 1].size();

return true;

}

#if 0

bool parseByRecursiveDescent(LexemInfo\* lexemInfoTable, Grammar\* grammar) {

int lexemIndex = 0;

const struct LexemInfo\* unexpectedLexemfailedTerminal = nullptr;

if (recursiveDescentParserRuleWithDebug(grammar->start\_symbol, lexemIndex, lexemInfoTable, grammar, 0, &unexpectedLexemfailedTerminal)) {

if (lexemInfoTable[lexemIndex].lexemStr[0] == '\0') {

printf("Parse successful.\n");

printf("%d.\n", lexemIndex);

return true;

}

else {

printf("Parse failed: Extra tokens remain.\n");

exit(0);

}

}

else {

if (unexpectedLexemfailedTerminal) {

printf("Parse failed in line.\r\n");

printf(" (The predicted terminal does not match the expected one.\r\n Possible unexpected terminal \"%s\" on line %lld at position %lld\r\n ..., but this is not certain.)\r\n", unexpectedLexemfailedTerminal->lexemStr, unexpectedLexemfailedTerminal->row, unexpectedLexemfailedTerminal->col);

}

else {

printf("Parse failed: unexpected terminal.\r\n");

}

exit(0);

}

return false;

}

bool parseByRecursiveDescent\_(LexemInfo\* lexemInfoTable, Grammar\* grammar) {

int lexemIndex = 0;

const struct LexemInfo unexpectedUnknownLexemfailedTerminal("unknown", 0, 0, 0, ~0, ~0); //

const struct LexemInfo\* returnUnexpectedLexemfailedTerminal = nullptr;

if (nullptr == (returnUnexpectedLexemfailedTerminal = recursiveDescentParserWithDebug\_(grammar->start\_symbol, lexemIndex, lexemInfoTable, grammar, 0, &unexpectedUnknownLexemfailedTerminal))) {

if (lexemInfoTable[lexemIndex].lexemStr[0] == '\0') {

printf("Parse successful.\n");

printf("%d.\n", lexemIndex);

return true;

}

else {

printf("Parse failed: Extra tokens remain.\n");

exit(0);

}

}

else {

if (returnUnexpectedLexemfailedTerminal->lexemStr[1]) {

printf("Parse failed.\r\n");

printf(" (The predicted terminal does not match the expected one.\r\n Possible unexpected terminal \"%s\" on line %lld at position %lld\r\n ..., but this is not certain.)\r\n", returnUnexpectedLexemfailedTerminal->lexemStr, returnUnexpectedLexemfailedTerminal->row, returnUnexpectedLexemfailedTerminal->col);

}

else {

printf("Parse failed: unexpected terminal.\r\n");

}

exit(0);

}

return false;

}

#endif  
  
while.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: while.cpp \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#include "stdio.h"

#include "string.h"

unsigned char\* makeWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_WHILE);

if (multitokenSize) {

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = \*\*lastLexemInfoInTable;

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned long long int)currBytePtr;

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

//

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@WHILE\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //\"%s\"\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

//

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_WHILE\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeNullStatementWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_NULL\_STATEMENT);

if (multitokenSize) {

if (lexemInfoTransformationTempStackSize < 2

|| strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

|| strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

|| lexemInfoTransformationTempStackSize >= 4 && !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

|| lexemInfoTransformationTempStackSize >= 3 && !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

return currBytePtr;

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_cmp\_eax\_0[] = { 0x83, 0xF8, 0x00 };

const unsigned char code\_\_jz\_offset[] = { 0x0F, 0x84, 0x00, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_cmp\_eax\_0, 3);

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jz\_offset, 6);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;after cond expresion (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //after cond expresion (after \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned long long int)(currBytePtr - 4);

//lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1];

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++ - 1].lexemStr, MAX\_LEXEM\_SIZE);

//lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++] = lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1];

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize++ - 1].lexemStr, MAX\_LEXEM\_SIZE);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, " cmp eax, 0\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jz LABEL@AFTER\_WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " if (opTemp == 0) goto LABEL\_\_AFTER\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}

unsigned char\* makeContinueWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_CONTINUE\_WHILE);

if (multitokenSize) {

if (

lexemInfoTransformationTempStackSize >= 6

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_THEN][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_IF][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 6].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 }; // jmp

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;continue while (in \"then\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //continue while (in \"then\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

//lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].ifvalue = (unsigned long long int)(currBytePtr - 4);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)((unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 6].ifvalue - currBytePtr);

}

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr, tokenStruct[MULTI\_TOKEN\_CONTINUE\_WHILE][0], MAX\_LEXEM\_SIZE);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 6].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 6].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

else if (

lexemInfoTransformationTempStackSize >= 5

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_ELSE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 }; // jmp

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;continue while (in \"else\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //continue while (in \"else\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

//lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].ifvalue = (unsigned long long int)(currBytePtr - 4);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)((unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].ifvalue - currBytePtr);

}

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr, tokenStruct[MULTI\_TOKEN\_CONTINUE\_WHILE][0], MAX\_LEXEM\_SIZE);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

else if (lexemInfoTransformationTempStackSize >= 4

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 }; // jmp

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;continue while (in \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //continue while (in \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

//lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned long long int)(currBytePtr - 4);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)((unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].ifvalue - currBytePtr);

}

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_CONTINUE\_WHILE][0], MAX\_LEXEM\_SIZE);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

}

return currBytePtr;

}

unsigned char\* makeExitWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) {

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_EXIT\_WHILE);

if (multitokenSize) {

if (

lexemInfoTransformationTempStackSize >= 6

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_THEN][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_IF][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 6].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 }; // jmp

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;exit while (in \"then\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //exit while (in \"then\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].ifvalue = (unsigned long long int)(currBytePtr - 4);

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr, tokenStruct[MULTI\_TOKEN\_EXIT\_WHILE][0], MAX\_LEXEM\_SIZE);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@AFTER\_WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_AFTER\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

else if (

lexemInfoTransformationTempStackSize >= 5

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_ELSE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 5].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 }; // jmp

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;exit while (in \"else\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //exit while (in \"else\"-part of %s-operator)\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned long long int)(currBytePtr - 4);

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_EXIT\_WHILE][0], MAX\_LEXEM\_SIZE);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@AFTER\_WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_AFTER\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

else if (lexemInfoTransformationTempStackSize >= 4

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 }; // jmp

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;exit while (in \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //exit while (in \"%s\")\r\n", tokenStruct[MULTI\_TOKEN\_WHILE][0]);

}

lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned long long int)(currBytePtr - 4);

strncpy(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_EXIT\_WHILE][0], MAX\_LEXEM\_SIZE);

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@AFTER\_WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_AFTER\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr);

}

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

}

return currBytePtr;

}

unsigned char\* makePostWhileCode\_(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode, unsigned char depthOfСontext) {

// if (!strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].lexemStr, tokenStruct[MULTI\_TOKEN\_CONTINUE\_WHILE][0], MAX\_LEXEM\_SIZE)) {

// \*(unsigned int\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue = (unsigned int)((unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 2].ifvalue - currBytePtr - 4);

// }

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

const unsigned char code\_\_jmp\_offset[] = { 0xE9, 0x00, 0x00, 0x00, 0x00 };

currBytePtr = outBytes2Code(currBytePtr, (unsigned char\*)code\_\_jmp\_offset, 5);

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == C\_CODER\_MODE) {

//

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

\*(unsigned int\*)(currBytePtr - 4) = (unsigned int)((unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].ifvalue - currBytePtr);

\*(unsigned int\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].ifvalue = (unsigned int)(currBytePtr - (unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].ifvalue - 4);

if (!strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].lexemStr, tokenStruct[MULTI\_TOKEN\_EXIT\_WHILE][0], MAX\_LEXEM\_SIZE)) {

\*(unsigned int\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue = (unsigned int)(currBytePtr - (unsigned char\*)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 1].ifvalue - 4);

}

}

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " jmp LABEL@WHILE\_%016llX\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr);

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL@AFTER\_WHILE\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr);

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += snprintf((char\*)currBytePtr, 8192, " goto LABEL\_\_WHILE\_%016llX;\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr);

currBytePtr += snprintf((char\*)currBytePtr, 8192, " LABEL\_\_AFTER\_WHILE\_%016llX:\r\n", (unsigned long long int)lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr);

}

return currBytePtr;

}

unsigned char\* makeEndWhileAfterWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode) { // Or Ender!

unsigned char multitokenSize = detectMultiToken(\*lastLexemInfoInTable, MULTI\_TOKEN\_END\_WHILE);

if (multitokenSize

&& lexemInfoTransformationTempStackSize >= 4

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 3].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

&& !strncmp(lexemInfoTransformationTempStack[lexemInfoTransformationTempStackSize - 4].lexemStr, tokenStruct[MULTI\_TOKEN\_WHILE][0], MAX\_LEXEM\_SIZE)

) {

if (generatorMode == MACHINE\_X86\_WIN32\_CODER\_MODE) {

//

}

else if (generatorMode == ASSEMBLY\_X86\_WIN32\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " ;end of while\r\n");

}

else if (generatorMode == C\_CODER\_MODE) {

currBytePtr += sprintf((char\*)currBytePtr, "\r\n");

currBytePtr += snprintf((char\*)currBytePtr, 8192, " //end of while\r\n");

}

currBytePtr = makePostWhileCode\_(lastLexemInfoInTable, currBytePtr, generatorMode, 0);

lexemInfoTransformationTempStackSize -= 4;

return \*lastLexemInfoInTable += multitokenSize, currBytePtr;

}

return currBytePtr;

}  
  
  
add.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: add.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define ADD\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeAddCode(B, C, M);

unsigned char\* makeAddCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
and.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: and.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define AND\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeAndCode(B, C, M);

unsigned char\* makeAndCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
cli.cpp  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: cw\_lex.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define PATH\_NAME\_LENGH 2048

#define MAX\_PARAMETERS\_SIZE 4096

#define PARAMETERS\_COUNT 32

//#define INPUT\_FILENAME\_PARAMETER 0

#define INPUT\_FILENAME\_WITH\_EXTENSION\_PARAMETER 1

#define OUT\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER 2

#define OUT\_LEXEME\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER 3

#define OUT\_AST\_FILENAME\_WITH\_EXTENSION\_PARAMETER 4

#define OUT\_SYNTAX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER 5

#define OUT\_SEMANTIX\_ERROR\_FILENAME\_WITH\_EXTENSION\_PARAMETER 6

#define OUT\_PREPARED\_LEXEMES\_SEQUENSE\_FILENAME\_WITH\_EXTENSION\_PARAMETER 7

#define OUT\_C\_FILENAME\_WITH\_EXTENSION\_PARAMETER 8

#define OUT\_ASSEMBLY\_FILENAME\_WITH\_EXTENSION\_PARAMETER 9

#define OUT\_OBJECT\_FILENAME\_WITH\_EXTENSION\_PARAMETER 10

#define OUT\_BINARY\_FILENAME\_WITH\_EXTENSION\_PARAMETER 11

#include "../../../src/include/def.h"

#include "../../../src/include/config.h"

#include "../../../src/include/generator/generator.h"

#include "../../../src/include/lexica/lexica.h"

//#include "stdio.h"

//#include "stdlib.h"

//#include "string.h"

#define DEFAULT\_INPUT\_FILENAME "../test\_programs/file1.cwl" // TODO: move!

extern unsigned long long int mode;

extern char parameters[PARAMETERS\_COUNT][MAX\_PARAMETERS\_SIZE];

void comandLineParser(int argc, char\* argv[], unsigned long long int\* mode, char(\*parameters)[MAX\_PARAMETERS\_SIZE]);

// after using this function use free(void \*) function to release text buffer

size\_t loadSource(char\*\* text, char\* fileName);  
  
config.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: config.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../include/def.h"

//#define LEXICAL\_ANALISIS\_MODE 1

//#define SEMANTIC\_ANALISIS\_MODE 2

//#define FULL\_COMPILER\_MODE 4

//#define DEBUG\_MODE 512

//#define DEFAULT\_MODE (DEBUG\_MODE | LEXICAL\_ANALISIS\_MODE)

//#define DEFAULT\_MODE (DEBUG\_MODE | LEXICAL\_ANALISIS\_MODE | SYNTAX\_ANALISIS\_MODE | SEMANTIC\_ANALISIS\_MODE | MAKE\_ASSEMBLY | MAKE\_BINARY)

#define TOKENS\_RE ";|<-|\\+\\+|--|\\\*\\\*|,|Eq|Neq|:|\\(|\\)|Ls|Gr|[\_0-9A-Za-z]+|[^ \t\r\f\v\n]"

#define KEYWORDS\_RE ";|<-|\\+\\+|--|\\\*\\\*|,|Eq|Neq|:|\\(|\\)|StartProgram|Variable|StartBlok|EndBlok|Exit|Continue|Scan|Print|If|else|For|To|Downto|Do|While|Repeat|Until|Goto|Div|Mod|<=|>=|Not|And|Or|Int32"

#define IDENTIFIERS\_RE "\_[a-z]"

#define UNSIGNEDVALUES\_RE "0|[1-9][0-9]\*"

// first column of the cw term paper option

#define PROGRAM\_FORMAT \

{"tokenNAME\_\_program\_name", 2, {"tokenNAME","program\_name"}},\

{"tokenSEMICOLON\_\_tokenBODY", 2, {"tokenSEMICOLON","tokenBODY"}},\

{"tokenDATA\_\_declaration", 2, {"tokenDATA","declaration"}},\

{"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY", 2, {"tokenNAME\_\_program\_name","tokenSEMICOLON\_\_tokenBODY"}},\

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA\_\_declaration"}},\

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA"}},\

{"statement\_\_tokenEND", 2, {"statement","tokenEND"}},\

{"statement\_\_\_\_iteration\_after\_two\_\_tokenEND", 2, {"statement\_\_\_\_iteration\_after\_two","tokenEND"}},\

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_\_\_iteration\_after\_two\_\_tokenEND"}},\

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_tokenEND"}},\

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","tokenEND"}},\

{"program", 2, {"program\_\_\_\_part1","program\_\_\_\_part2"}},

#define T\_NAME\_0 "StartProgram"

#define T\_NAME\_1 ""

#define T\_NAME\_2 ""

#define T\_NAME\_3 ""

#define T\_BODY\_0 "StartBlok"

#define T\_BODY\_1 ""

#define T\_BODY\_2 ""

#define T\_BODY\_3 ""

#define T\_DATA\_0 "Variable"

#define T\_DATA\_1 ""

#define T\_DATA\_2 ""

#define T\_DATA\_3 ""

#define T\_DATA\_TYPE\_0 "Int32"

#define T\_DATA\_TYPE\_1 ""

#define T\_DATA\_TYPE\_2 ""

#define T\_DATA\_TYPE\_3 ""

//

#define T\_BITWISE\_NOT\_0 ""

#define T\_BITWISE\_NOT\_1 ""

#define T\_BITWISE\_NOT\_2 ""

#define T\_BITWISE\_NOT\_3 ""

#define T\_BITWISE\_AND\_0 ""

#define T\_BITWISE\_AND\_1 ""

#define T\_BITWISE\_AND\_2 ""

#define T\_BITWISE\_AND\_3 ""

#define T\_BITWISE\_OR\_0 ""

#define T\_BITWISE\_OR\_1 ""

#define T\_BITWISE\_OR\_2 ""

#define T\_BITWISE\_OR\_3 ""

#define T\_NOT\_0 "Not"

#define T\_NOT\_1 ""

#define T\_NOT\_2 ""

#define T\_NOT\_3 ""

#define T\_AND\_0 "And"

#define T\_AND\_1 ""

#define T\_AND\_2 ""

#define T\_AND\_3 ""

#define T\_OR\_0 "Or"

#define T\_OR\_1 ""

#define T\_OR\_2 ""

#define T\_OR\_3 ""

//

#define T\_EQUAL\_0 "Eq"

#define T\_EQUAL\_1 ""

#define T\_EQUAL\_2 ""

#define T\_EQUAL\_3 ""

#define T\_NOT\_EQUAL\_0 "Neq"

#define T\_NOT\_EQUAL\_1 ""

#define T\_NOT\_EQUAL\_2 ""

#define T\_NOT\_EQUAL\_3 ""

#define T\_LESS\_0 "Ls"

#define T\_LESS\_1 ""

#define T\_LESS\_2 ""

#define T\_LESS\_3 ""

#define T\_GREATER\_0 "Gr"

#define T\_GREATER\_1 ""

#define T\_GREATER\_2 ""

#define T\_GREATER\_3 ""

#define T\_LESS\_OR\_EQUAL\_0 ""

#define T\_LESS\_OR\_EQUAL\_1 ""

#define T\_LESS\_OR\_EQUAL\_2 ""

#define T\_LESS\_OR\_EQUAL\_3 ""

#define T\_GREATER\_OR\_EQUAL\_0 ""

#define T\_GREATER\_OR\_EQUAL\_1 ""

#define T\_GREATER\_OR\_EQUAL\_2 ""

#define T\_GREATER\_OR\_EQUAL\_3 ""

//

#define T\_ADD\_0 "++"

#define T\_ADD\_1 ""

#define T\_ADD\_2 ""

#define T\_ADD\_3 ""

#define T\_SUB\_0 "--"

#define T\_SUB\_1 ""

#define T\_SUB\_2 ""

#define T\_SUB\_3 ""

#define T\_MUL\_0 "\*\*"

#define T\_MUL\_1 ""

#define T\_MUL\_2 ""

#define T\_MUL\_3 ""

#define T\_DIV\_0 "Div"

#define T\_DIV\_1 ""

#define T\_DIV\_2 ""

#define T\_DIV\_3 ""

#define T\_MOD\_0 "Mod"

#define T\_MOD\_1 ""

#define T\_MOD\_2 ""

#define T\_MOD\_3 ""

//

#define T\_BIND\_RIGHT\_TO\_LEFT\_0 "<-"

#define T\_BIND\_RIGHT\_TO\_LEFT\_1 ""

#define T\_BIND\_RIGHT\_TO\_LEFT\_2 ""

#define T\_BIND\_RIGHT\_TO\_LEFT\_3 ""

#define T\_BIND\_LEFT\_TO\_RIGHT\_0 ""

#define T\_BIND\_LEFT\_TO\_RIGHT\_1 ""

#define T\_BIND\_LEFT\_TO\_RIGHT\_2 ""

#define T\_BIND\_LEFT\_TO\_RIGHT\_3 ""

//

#define T\_COMA\_0 ","

#define T\_COMA\_1 ""

#define T\_COMA\_2 ""

#define T\_COMA\_3 ""

#define T\_COLON\_0 ":"

#define T\_COLON\_1 ""

#define T\_COLON\_2 ""

#define T\_COLON\_3 ""

#define T\_GOTO\_0 "Goto"

#define T\_GOTO\_1 ""

#define T\_GOTO\_2 ""

#define T\_GOTO\_3 ""

//

#define T\_IF\_0 "If"

#define T\_IF\_1 "("

#define T\_IF\_2 ""

#define T\_IF\_3 ""

#define T\_THEN\_0 ")"

#define T\_THEN\_1 ""

#define T\_THEN\_2 ""

#define T\_THEN\_3 ""

#define T\_ELSE\_0 "else"

#define T\_ELSE\_1 ""

#define T\_ELSE\_2 ""

#define T\_ELSE\_3 ""

//

#define T\_FOR\_0 "For"

#define T\_FOR\_1 ""

#define T\_FOR\_2 ""

#define T\_FOR\_3 ""

#define T\_TO\_0 "To"

#define T\_TO\_1 ""

#define T\_TO\_2 ""

#define T\_TO\_3 ""

#define T\_DOWNTO\_0 "Downto"

#define T\_DOWNTO\_1 ""

#define T\_DOWNTO\_2 ""

#define T\_DOWNTO\_3 ""

#define T\_DO\_0 "Do"

#define T\_DO\_1 ""

#define T\_DO\_2 ""

#define T\_DO\_3 ""

//

#define T\_WHILE\_0 "While"

#define T\_WHILE\_1 ""

#define T\_WHILE\_2 ""

#define T\_WHILE\_3 ""

#define T\_CONTINUE\_WHILE\_0 "Continue"

#define T\_CONTINUE\_WHILE\_1 "While"

#define T\_CONTINUE\_WHILE\_2 ""

#define T\_CONTINUE\_WHILE\_3 ""

#define T\_EXIT\_WHILE\_0 "Exit"

#define T\_EXIT\_WHILE\_1 "While"

#define T\_EXIT\_WHILE\_2 ""

#define T\_EXIT\_WHILE\_3 ""

#define T\_END\_WHILE\_0 "End"

#define T\_END\_WHILE\_1 "While"

#define T\_END\_WHILE\_2 ""

#define T\_END\_WHILE\_3 ""

//

#define T\_REPEAT\_0 "Repeat"

#define T\_REPEAT\_1 ""

#define T\_REPEAT\_2 ""

#define T\_REPEAT\_3 ""

#define T\_UNTIL\_0 "Until"

#define T\_UNTIL\_1 ""

#define T\_UNTIL\_2 ""

#define T\_UNTIL\_3 ""

//

#define T\_INPUT\_0 "Scan"

#define T\_INPUT\_1 ""

#define T\_INPUT\_2 ""

#define T\_INPUT\_3 ""

#define T\_OUTPUT\_0 "Print"

#define T\_OUTPUT\_1 ""

#define T\_OUTPUT\_2 ""

#define T\_OUTPUT\_3 ""

//

#define T\_RLBIND\_0 "<-"

#define T\_RLBIND\_1 ""

#define T\_RLBIND\_2 ""

#define T\_RLBIND\_3 ""

#define T\_LRBIND\_0 ""

#define T\_LRBIND\_1 ""

#define T\_LRBIND\_2 ""

#define T\_LRBIND\_3 ""

//

#define T\_SEMICOLON\_0 ";"

#define T\_SEMICOLON\_1 ""

#define T\_SEMICOLON\_2 ""

#define T\_SEMICOLON\_3 ""

//

#define T\_BEGIN\_0 "StartBlok"

#define T\_BEGIN\_1 ""

#define T\_BEGIN\_2 ""

#define T\_BEGIN\_3 ""

#define T\_END\_0 "EndBlok"

#define T\_END\_1 ""

#define T\_END\_2 ""

#define T\_END\_3 ""

//

#define T\_NULL\_STATEMENT\_0 "Null"

#define T\_NULL\_STATEMENT\_1 "Statement"

#define T\_NULL\_STATEMENT\_2 ""

#define T\_NULL\_STATEMENT\_3 ""

#ifndef TOKEN\_STRUCT\_NAME\_

#define TOKEN\_STRUCT\_NAME\_

DECLENUM(TokenStructName,

MULTI\_TOKEN\_BITWISE\_NOT,

MULTI\_TOKEN\_BITWISE\_AND,

MULTI\_TOKEN\_BITWISE\_OR,

MULTI\_TOKEN\_NOT,

MULTI\_TOKEN\_AND,

MULTI\_TOKEN\_OR,

MULTI\_TOKEN\_EQUAL,

MULTI\_TOKEN\_NOT\_EQUAL,

MULTI\_TOKEN\_LESS,

MULTI\_TOKEN\_GREATER,

MULTI\_TOKEN\_LESS\_OR\_EQUAL,

MULTI\_TOKEN\_GREATER\_OR\_EQUAL,

MULTI\_TOKEN\_ADD,

MULTI\_TOKEN\_SUB,

MULTI\_TOKEN\_MUL,

MULTI\_TOKEN\_DIV,

MULTI\_TOKEN\_MOD,

MULTI\_TOKEN\_BIND\_RIGHT\_TO\_LEFT,

MULTI\_TOKEN\_BIND\_LEFT\_TO\_RIGHT,

MULTI\_TOKEN\_COLON,

MULTI\_TOKEN\_GOTO,

MULTI\_TOKEN\_IF,

// MULTI\_TOKEN\_IF\_, // don't change this!

MULTI\_TOKEN\_THEN,

// MULTI\_TOKEN\_THEN\_, // don't change this!

MULTI\_TOKEN\_ELSE,

MULTI\_TOKEN\_FOR,

MULTI\_TOKEN\_TO,

MULTI\_TOKEN\_DOWNTO,

MULTI\_TOKEN\_DO,

//

MULTI\_TOKEN\_WHILE,

/\*while special statement\*/MULTI\_TOKEN\_CONTINUE\_WHILE,

/\*while special statement\*/MULTI\_TOKEN\_EXIT\_WHILE,

MULTI\_TOKEN\_END\_WHILE,

//

//

MULTI\_TOKEN\_REPEAT,

MULTI\_TOKEN\_UNTIL,

//

//

MULTI\_TOKEN\_INPUT,

MULTI\_TOKEN\_OUTPUT,

//

//

MULTI\_TOKEN\_RLBIND,

MULTI\_TOKEN\_LRBIND,

//

MULTI\_TOKEN\_SEMICOLON,

MULTI\_TOKEN\_BEGIN,

MULTI\_TOKEN\_END,

//

MULTI\_TOKEN\_NULL\_STATEMENT

);

//#define PROCESS\_TOKENS(...) HANDLE\_TOKENS(\_\_VA\_ARGS\_\_)

//#define TOKENS\_FOR\_MULTI\_TOKEN(A, B, C, D) A, B, C, D

//#define TOKENS\_FOR\_MULTI\_TOKEN\_BITWISE\_NOT TOKENS\_FOR\_MULTI\_TOKEN("~", "", "", "")

#define INIT\_TOKEN\_STRUCT\_NAME() static void intitTokenStruct(){\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, BITWISE\_NOT)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, BITWISE\_AND)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, BITWISE\_OR)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, NOT)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, AND)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, OR)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, EQUAL)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, NOT\_EQUAL)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, LESS)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, GREATER)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, LESS\_OR\_EQUAL)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, GREATER\_OR\_EQUAL)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, ADD)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, SUB)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, MUL)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, DIV)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, MOD)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, BIND\_RIGHT\_TO\_LEFT)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, BIND\_LEFT\_TO\_RIGHT)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, COLON)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, GOTO)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, IF)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, THEN)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, ELSE)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, FOR)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, TO)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, DOWNTO)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, DO)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, WHILE)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, CONTINUE\_WHILE)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, EXIT\_WHILE)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, END\_WHILE)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, REPEAT)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, UNTIL)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, INPUT)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, OUTPUT)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, RLBIND)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, LRBIND)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, SEMICOLON)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, BEGIN)\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, END)\

\

SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(tokenStruct, NULL\_STATEMENT)\

} char intitTokenStruct\_ = (intitTokenStruct(), 0);

#define MAX\_TOKEN\_STRUCT\_ELEMENT\_COUNT GET\_ENUM\_SIZE(TokenStructName)

#define MAX\_TOKEN\_STRUCT\_ELEMENT\_PART\_COUNT 4

#endif

extern char\* tokenStruct[MAX\_TOKEN\_STRUCT\_ELEMENT\_COUNT][MAX\_TOKEN\_STRUCT\_ELEMENT\_PART\_COUNT];

#define CONFIGURABLE\_GRAMMAR {\

{"labeled\_point", 2, {"ident", "tokenCOLON"}},\

{"goto\_label", 2, {"tokenGOTO","ident"}},\

{"program\_name", 1, {"ident\_terminal"}},\

{"value\_type", 1, {T\_DATA\_TYPE\_0}},\

{"other\_declaration\_ident", 2, {"tokenCOMMA", "ident"}},\

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"other\_declaration\_ident","other\_declaration\_ident\_\_\_\_iteration\_after\_one"}},\

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"tokenCOMMA", "ident"}},\

{"value\_type\_\_ident", 2, {"value\_type", "ident"}},\

{"declaration", 2, {"value\_type\_\_ident", "other\_declaration\_ident\_\_\_\_iteration\_after\_one"}},\

{"declaration", 2, {"value\_type", "ident"}},\

\

{"unary\_operator", 1, {T\_NOT\_0}},\

{"unary\_operator", 1, {T\_SUB\_0}},\

{"unary\_operator", 1, {T\_ADD\_0}},\

{"binary\_operator", 1, {T\_AND\_0}},\

{"binary\_operator", 1, {T\_OR\_0}},\

{"binary\_operator", 1, {T\_EQUAL\_0}},\

{"binary\_operator", 1, {T\_NOT\_EQUAL\_0}},\

{"binary\_operator", 1, {T\_LESS\_OR\_EQUAL\_0}},\

{"binary\_operator", 1, {T\_GREATER\_OR\_EQUAL\_0}},\

{"binary\_operator", 1, {T\_ADD\_0}},\

{"binary\_operator", 1, {T\_SUB\_0}},\

{"binary\_operator", 1, {T\_MUL\_0}},\

{"binary\_operator", 1, {T\_DIV\_0}},\

{"binary\_operator", 1, {T\_MOD\_0}},\

{"binary\_action", 2, {"binary\_operator","expression"}},\

\

{"left\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},\

{"left\_expression", 2, {"unary\_operator","expression"}},\

{"left\_expression", 1, {"ident\_terminal"}},\

{"left\_expression", 1, {"value\_terminal"}},\

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action\_\_\_\_iteration\_after\_two"}},\

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action"}},\

{"expression", 2, {"left\_expression","binary\_action\_\_\_\_iteration\_after\_two"}},\

{"expression", 2, {"left\_expression","binary\_action"}},\

{"expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},\

{"expression", 2, {"unary\_operator","expression"}},\

{"expression", 1, {"ident\_terminal"}},\

{"expression", 1, {"value\_terminal"}},\

\

{"tokenGROUPEXPRESSIONBEGIN\_\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN","expression"}},\

{"group\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},\

\

{"bind\_right\_to\_left", 2, {"ident","rl\_expression"}},\

{"bind\_left\_to\_right", 2, {"lr\_expression","ident"}},\

\

{"body\_for\_true", 2, {"statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},\

{"body\_for\_true", 2, {"statement\_in\_while\_body","tokenSEMICOLON"}},\

{"body\_for\_true", 1, {T\_SEMICOLON\_0}},\

{"tokenELSE\_\_statement\_in\_while\_body", 2, {"tokenELSE","statement\_in\_while\_body"}},\

{"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenELSE","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},\

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},\

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body","tokenSEMICOLON"}},\

{"body\_for\_false", 2, {"tokenELSE","tokenSEMICOLON"}},\

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN", 2, {"tokenIF","tokenGROUPEXPRESSIONBEGIN"}},\

{"expression\_\_tokenGROUPEXPRESSIONEND", 2, {"expression","tokenGROUPEXPRESSIONEND"}},\

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN","expression\_\_tokenGROUPEXPRESSIONEND"}},\

{"body\_for\_true\_\_body\_for\_false", 2, {"body\_for\_true","body\_for\_false"}},\

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},\

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},\

\

{"cycle\_counter", 1, {"ident\_terminal"}},\

{"rl\_expression", 2, {"tokenRLBIND","expression"}},\

{"lr\_expression", 2, {"expression","tokenLRBIND"}},\

{"cycle\_counter\_init", 2, {"cycle\_counter","rl\_expression"}},\

{"cycle\_counter\_init", 2, {"lr\_expression","cycle\_counter"}},\

{"cycle\_counter\_last\_value", 1, {"value\_terminal"}},\

{"cycle\_body", 2, {"tokenDO","statement\_\_\_\_iteration\_after\_two"}},\

{"cycle\_body", 2, {"tokenDO","statement"}},\

{"tokenFOR\_\_cycle\_counter\_init", 2, {"tokenFOR","cycle\_counter\_init"}},\

{"tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenTO","cycle\_counter\_last\_value"}},\

{"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenFOR\_\_cycle\_counter\_init","tokenTO\_\_cycle\_counter\_last\_value"}},\

{"cycle\_body\_\_tokenSEMICOLON", 2, {"cycle\_body","tokenSEMICOLON"}},\

{"forto\_cycle", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},\

\

{"continue\_while", 2, {"tokenCONTINUE","tokenWHILE"}},\

{"exit\_while", 2, {"tokenEXIT","tokenWHILE"}},\

{"tokenWHILE\_\_expression", 2, {"tokenWHILE","expression"}},\

{"tokenEND\_\_tokenWHILE", 2, {"tokenENDWHILE\_END","tokenENDWHILE\_WHILE"}},\

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body"}},\

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},\

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},\

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},\

{"while\_cycle", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},\

\

{"tokenUNTIL\_\_expression", 2, {"tokenUNTIL","expression"}},\

{"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two", 2, {"tokenREPEAT","statement\_\_\_\_iteration\_after\_two"}},\

{"tokenREPEAT\_\_statement", 2, {"tokenREPEAT","statement"}},\

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},\

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},\

{"repeat\_until\_cycle", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},\

\

{"input\_\_first\_part", 2, {"tokenGET","tokenGROUPEXPRESSIONBEGIN"}},\

{"input\_\_second\_part", 2, {"ident","tokenGROUPEXPRESSIONEND"}},\

{"input", 2, {"input\_\_first\_part","input\_\_second\_part"}},\

\

{"output\_\_first\_part", 2, {"tokenPUT","tokenGROUPEXPRESSIONBEGIN"}},\

{"output\_\_second\_part", 2, {"expression","tokenGROUPEXPRESSIONEND"}},\

{"output", 2, {"output\_\_first\_part","output\_\_second\_part"}},\

\

{"statement", 2, {"ident","rl\_expression"}},\

{"statement", 2, {"lr\_expression","ident"}},\

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},\

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},\

{"statement", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},\

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},\

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},\

{"statement", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},\

{"statement", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},\

{"statement", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},\

{"statement", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},\

{"statement", 2, {"ident","tokenCOLON"}},\

{"statement", 2, {"tokenGOTO","ident"}},\

{"statement", 2, {"input\_\_first\_part","input\_\_second\_part"}},\

{"statement", 2, {"output\_\_first\_part","output\_\_second\_part"}},\

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement\_\_\_\_iteration\_after\_two"}},\

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement"}},\

\

{ "statement\_in\_while\_body", 2, {"ident","rl\_expression"}},\

{ "statement\_in\_while\_body", 2, {"lr\_expression","ident"}},\

{ "statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},\

{ "statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},\

{ "statement\_in\_while\_body", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},\

{ "statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},\

{ "statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},\

{ "statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},\

{ "statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},\

{ "statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},\

{ "statement\_in\_while\_body", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},\

{ "statement\_in\_while\_body", 2, {"ident","tokenCOLON"}},\

{ "statement\_in\_while\_body", 2, {"tokenGOTO","ident"}},\

{ "statement\_in\_while\_body", 2, {"input\_\_first\_part","input\_\_second\_part"}},\

{ "statement\_in\_while\_body", 2, {"output\_\_first\_part","output\_\_second\_part"}},\

{ "statement\_in\_while\_body", 2, {"tokenCONTINUE","tokenWHILE"}},\

{ "statement\_in\_while\_body", 2, {"tokenEXIT","tokenWHILE"}},\

{ "statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},\

{ "statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body"}},\

\

PROGRAM\_FORMAT\

\

{"tokenCOLON", 1, {T\_COLON\_0}},\

{"tokenGOTO", 1, {T\_GOTO\_0}},\

{"tokenINTEGER16", 1, {T\_DATA\_TYPE\_0}},\

{"tokenCOMMA", 1, {T\_COMA\_0}},\

{"tokenNOT", 1, {T\_NOT\_0}},\

{"tokenAND", 1, {T\_AND\_0}},\

{"tokenOR", 1, {T\_OR\_0}},\

{"tokenEQUAL", 1, {T\_EQUAL\_0}},\

{"tokenNOTEQUAL", 1, {T\_NOT\_EQUAL\_0}},\

{"tokenLESSOREQUAL", 1, {T\_LESS\_OR\_EQUAL\_0}},\

{"tokenGREATEROREQUAL", 1, {T\_GREATER\_OR\_EQUAL\_0}},\

{"tokenPLUS", 1, {T\_ADD\_0}},\

{"tokenMINUS", 1, {T\_SUB\_0}},\

{"tokenMUL", 1, {T\_MUL\_0}},\

{"tokenDIV", 1, {T\_DIV\_0}},\

{"tokenMOD", 1, {T\_MOD\_0}},\

{"tokenGROUPEXPRESSIONBEGIN", 1, {"("}},\

{"tokenGROUPEXPRESSIONEND", 1, {")"}},\

{"tokenRLBIND", 1, {T\_RLBIND\_0}},\

{"tokenLRBIND", 1, {T\_LRBIND\_0}},\

{"tokenELSE", 1, {T\_ELSE\_0}},\

{"tokenIF", 1, {T\_IF\_0}},\

{"tokenDO", 1, {T\_DO\_0}},\

{"tokenFOR", 1, {T\_FOR\_0}},\

{"tokenTO", 1, {T\_TO\_0}},\

{"tokenWHILE", 1, {T\_WHILE\_0}},\

{"tokenCONTINUE", 1, {T\_CONTINUE\_WHILE\_0}},\

{"tokenEXIT", 1, {T\_EXIT\_WHILE\_0}},\

{"tokenENDWHILE\_END", 1, {T\_END\_WHILE\_0}},\

{"tokenENDWHILE\_WHILE", 1, {T\_END\_WHILE\_1}},\

{"tokenREPEAT", 1, {T\_REPEAT\_0}},\

{"tokenUNTIL", 1, {T\_UNTIL\_0}},\

{"tokenGET", 1, {T\_INPUT\_0}},\

{"tokenPUT", 1, {T\_OUTPUT\_0}},\

{"tokenNAME", 1, {T\_NAME\_0}},\

{"tokenBODY", 1, {T\_BODY\_0}},\

{"tokenDATA", 1, {T\_DATA\_0}},\

{"tokenEND", 1, {T\_END\_0}},\

{"tokenSEMICOLON", 1, {T\_SEMICOLON\_0}},\

\

{"value", 1, {"value\_terminal"}},\

\

{"ident", 1, {"ident\_terminal"}},\

\

{"", 2, {"",""}}\

},\

178,\

"program"

#define ORIGINAL\_GRAMMAR {\

{"labeled\_point", 2, {"ident", "tokenCOLON"}},\

{"goto\_label", 2, {"tokenGOTO","ident"}},\

{"program\_name", 1, {"ident\_terminal"}},\

{"value\_type", 1, {"INTEGER16"}},\

{"other\_declaration\_ident", 2, {"tokenCOMMA", "ident"}},\

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"other\_declaration\_ident","other\_declaration\_ident\_\_\_\_iteration\_after\_one"}},\

{"other\_declaration\_ident\_\_\_\_iteration\_after\_one", 2, {"tokenCOMMA", "ident"}},\

{"value\_type\_\_ident", 2, {"value\_type", "ident"}},\

{"declaration", 2, {"value\_type\_\_ident", "other\_declaration\_ident\_\_\_\_iteration\_after\_one"}},\

{"declaration", 2, {"value\_type", "ident"}},\

\

{"unary\_operator", 1, {"NOT"}},\

{"unary\_operator", 1, {"-"}},\

{"unary\_operator", 1, {"+"}},\

{"binary\_operator", 1, {"AND"}},\

{"binary\_operator", 1, {"OR"}},\

{"binary\_operator", 1, {"=="}},\

{"binary\_operator", 1, {"!="}},\

{"binary\_operator", 1, {"<="}},\

{"binary\_operator", 1, {">="}},\

{"binary\_operator", 1, {"+"}},\

{"binary\_operator", 1, {"-"}},\

{"binary\_operator", 1, {"\*"}},\

{"binary\_operator", 1, {"DIV"}},\

{"binary\_operator", 1, {"MOD"}},\

{"binary\_action", 2, {"binary\_operator","expression"}},\

\

{"left\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},\

{"left\_expression", 2, {"unary\_operator","expression"}},\

{"left\_expression", 1, {"ident\_terminal"}},\

{"left\_expression", 1, {"value\_terminal"}},\

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action\_\_\_\_iteration\_after\_two"}},\

{"binary\_action\_\_\_\_iteration\_after\_two", 2, {"binary\_action","binary\_action"}},\

{"expression", 2, {"left\_expression","binary\_action\_\_\_\_iteration\_after\_two"}},\

{"expression", 2, {"left\_expression","binary\_action"}},\

{"expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},\

{"expression", 2, {"unary\_operator","expression"}},\

{"expression", 1, {"ident\_terminal"}},\

{"expression", 1, {"value\_terminal"}},\

\

{"tokenGROUPEXPRESSIONBEGIN\_\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN","expression"}},\

{"group\_expression", 2, {"tokenGROUPEXPRESSIONBEGIN\_\_expression","tokenGROUPEXPRESSIONEND"}},\

\

{"bind\_right\_to\_left", 2, {"ident","rl\_expression"}},\

{"bind\_left\_to\_right", 2, {"lr\_expression","ident"}},\

\

{"body\_for\_true", 2, {"statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},\

{"body\_for\_true", 2, {"statement\_in\_while\_body","tokenSEMICOLON"}},\

{"body\_for\_true", 1, {";"}},\

{"tokenELSE\_\_statement\_in\_while\_body", 2, {"tokenELSE","statement\_in\_while\_body"}},\

{"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenELSE","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},\

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenSEMICOLON"}},\

{"body\_for\_false", 2, {"tokenELSE\_\_statement\_in\_while\_body","tokenSEMICOLON"}},\

{"body\_for\_false", 2, {"tokenELSE","tokenSEMICOLON"}},\

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN", 2, {"tokenIF","tokenGROUPEXPRESSIONBEGIN"}},\

{"expression\_\_tokenGROUPEXPRESSIONEND", 2, {"expression","tokenGROUPEXPRESSIONEND"}},\

{"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN","expression\_\_tokenGROUPEXPRESSIONEND"}},\

{"body\_for\_true\_\_body\_for\_false", 2, {"body\_for\_true","body\_for\_false"}},\

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},\

{"cond\_block", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},\

\

{"cycle\_counter", 1, {"ident\_terminal"}},\

{"rl\_expression", 2, {"tokenRLBIND","expression"}},\

{"lr\_expression", 2, {"expression","tokenLRBIND"}},\

{"cycle\_counter\_init", 2, {"cycle\_counter","rl\_expression"}},\

{"cycle\_counter\_init", 2, {"lr\_expression","cycle\_counter"}},\

{"cycle\_counter\_last\_value", 1, {"value\_terminal"}},\

{"cycle\_body", 2, {"tokenDO","statement\_\_\_\_iteration\_after\_two"}},\

{"cycle\_body", 2, {"tokenDO","statement"}},\

{"tokenFOR\_\_cycle\_counter\_init", 2, {"tokenFOR","cycle\_counter\_init"}},\

{"tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenTO","cycle\_counter\_last\_value"}},\

{"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value", 2, {"tokenFOR\_\_cycle\_counter\_init","tokenTO\_\_cycle\_counter\_last\_value"}},\

{"cycle\_body\_\_tokenSEMICOLON", 2, {"cycle\_body","tokenSEMICOLON"}},\

{"forto\_cycle", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},\

\

{"continue\_while", 2, {"tokenCONTINUE","tokenWHILE"}},\

{"exit\_while", 2, {"tokenEXIT","tokenWHILE"}},\

{"tokenWHILE\_\_expression", 2, {"tokenWHILE","expression"}},\

{"tokenEND\_\_tokenWHILE", 2, {"tokenEND","tokenWHILE"}},\

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body"}},\

{"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"tokenWHILE\_\_expression","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},\

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},\

{"while\_cycle", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},\

{"while\_cycle", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},\

\

{"tokenUNTIL\_\_expression", 2, {"tokenUNTIL","expression"}},\

{"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two", 2, {"tokenREPEAT","statement\_\_\_\_iteration\_after\_two"}},\

{"tokenREPEAT\_\_statement", 2, {"tokenREPEAT","statement"}},\

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},\

{"repeat\_until\_cycle", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},\

{"repeat\_until\_cycle", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},\

\

{"input\_\_first\_part", 2, {"tokenGET","tokenGROUPEXPRESSIONBEGIN"}},\

{"input\_\_second\_part", 2, {"ident","tokenGROUPEXPRESSIONEND"}},\

{"input", 2, {"input\_\_first\_part","input\_\_second\_part"}},\

\

{"output\_\_first\_part", 2, {"tokenPUT","tokenGROUPEXPRESSIONBEGIN"}},\

{"output\_\_second\_part", 2, {"expression","tokenGROUPEXPRESSIONEND"}},\

{"output", 2, {"output\_\_first\_part","output\_\_second\_part"}},\

\

{"statement", 2, {"ident","rl\_expression"}},\

{"statement", 2, {"lr\_expression","ident"}},\

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},\

{"statement", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},\

{"statement", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},\

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},\

{"statement", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},\

{"statement", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},\

{"statement", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},\

{"statement", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},\

{"statement", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},\

{"statement", 2, {"ident","tokenCOLON"}},\

{"statement", 2, {"tokenGOTO","ident"}},\

{"statement", 2, {"input\_\_first\_part","input\_\_second\_part"}},\

{"statement", 2, {"output\_\_first\_part","output\_\_second\_part"}},\

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement\_\_\_\_iteration\_after\_two"}},\

{"statement\_\_\_\_iteration\_after\_two", 2, {"statement","statement"}},\

\

{"statement\_in\_while\_body", 2, {"ident","rl\_expression"}},\

{"statement\_in\_while\_body", 2, {"lr\_expression","ident"}},\

{"statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true\_\_body\_for\_false"}},\

{"statement\_in\_while\_body", 2, {"tokenIF\_\_tokenGROUPEXPRESSIONBEGIN\_\_expression\_\_tokenGROUPEXPRESSIONEND","body\_for\_true"}},\

{"statement\_in\_while\_body", 2, {"tokenFOR\_\_cycle\_counter\_init\_\_tokenTO\_\_cycle\_counter\_last\_value","cycle\_body\_\_tokenSEMICOLON"}},\

{"statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body\_\_\_\_iteration\_after\_two","tokenEND\_\_tokenWHILE"}},\

{"statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression\_\_statement\_in\_while\_body","tokenEND\_\_tokenWHILE"}},\

{"statement\_in\_while\_body", 2, {"tokenWHILE\_\_expression","tokenEND\_\_tokenWHILE"}},\

{"statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement\_\_\_\_iteration\_after\_two","tokenUNTIL\_\_expression"}},\

{"statement\_in\_while\_body", 2, {"tokenREPEAT\_\_statement","tokenUNTIL\_\_expression"}},\

{"statement\_in\_while\_body", 2, {"tokenREPEAT","tokenUNTIL\_\_expression"}},\

{"statement\_in\_while\_body", 2, {"ident","tokenCOLON"}},\

{"statement\_in\_while\_body", 2, {"tokenGOTO","ident"}},\

{"statement\_in\_while\_body", 2, {"input\_\_first\_part","input\_\_second\_part"}},\

{"statement\_in\_while\_body", 2, {"output\_\_first\_part","output\_\_second\_part"}},\

{"statement\_in\_while\_body", 2, {"tokenCONTINUE","tokenWHILE"}},\

{"statement\_in\_while\_body", 2, {"tokenEXIT","tokenWHILE"}},\

{"statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body\_\_\_\_iteration\_after\_two"}},\

{"statement\_in\_while\_body\_\_\_\_iteration\_after\_two", 2, {"statement\_in\_while\_body","statement\_in\_while\_body"}},\

\

{"tokenNAME\_\_program\_name", 2, {"tokenNAME","program\_name"}},\

{"tokenSEMICOLON\_\_tokenBODY", 2, {"tokenSEMICOLON","tokenBODY"}},\

{"tokenDATA\_\_declaration", 2, {"tokenDATA","declaration"}},\

{"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY", 2, {"tokenNAME\_\_program\_name","tokenSEMICOLON\_\_tokenBODY"}},\

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA\_\_declaration"}},\

{"program\_\_\_\_part1", 2, {"tokenNAME\_\_program\_name\_\_tokenSEMICOLON\_\_tokenBODY","tokenDATA"}},\

{"statement\_\_tokenEND", 2, {"statement","tokenEND"}},\

{"statement\_\_\_\_iteration\_after\_two\_\_tokenEND", 2, {"statement\_\_\_\_iteration\_after\_two","tokenEND"}},\

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_\_\_iteration\_after\_two\_\_tokenEND"}},\

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","statement\_\_tokenEND"}},\

{"program\_\_\_\_part2", 2, {"tokenSEMICOLON","tokenEND"}},\

{"program", 2, {"program\_\_\_\_part1","program\_\_\_\_part2"}},\

\

{"tokenCOLON", 1, {":"}},\

{"tokenGOTO", 1, {"GOTO"}},\

{"tokenINTEGER16", 1, {"INTEGER16"}},\

{"tokenCOMMA", 1, {","}},\

{"tokenNOT", 1, {"NOT"}},\

{"tokenAND", 1, {"AND"}},\

{"tokenOR", 1, {"OR"}},\

{"tokenEQUAL", 1, {"=="}},\

{"tokenNOTEQUAL", 1, {"!="}},\

{"tokenLESSOREQUAL", 1, {"<="}},\

{"tokenGREATEROREQUAL", 1, {">="}},\

{"tokenPLUS", 1, {"+"}},\

{"tokenMINUS", 1, {"-"}},\

{"tokenMUL", 1, {"\*"}},\

{"tokenDIV", 1, {"DIV"}},\

{"tokenMOD", 1, {"MOD"}},\

{"tokenGROUPEXPRESSIONBEGIN", 1, {"("}},\

{"tokenGROUPEXPRESSIONEND", 1, {")"}},\

{"tokenRLBIND", 1, {"<<"}},\

{"tokenLRBIND", 1, {">>"}},\

{"tokenELSE", 1, {"ELSE"}},\

{"tokenIF", 1, {"IF"}},\

{"tokenDO", 1, {"DO"}},\

{"tokenFOR", 1, {"FOR"}},\

{"tokenTO", 1, {"TO"}},\

{"tokenWHILE", 1, {"WHILE"}},\

{"tokenCONTINUE", 1, {"CONTINUE"}},\

{"tokenEXIT", 1, {"EXIT"}},\

{"tokenREPEAT", 1, {"REPEAT"}},\

{"tokenUNTIL", 1, {"UNTIL"}},\

{"tokenGET", 1, {"GET"}},\

{"tokenPUT", 1, {"PUT"}},\

{"tokenNAME", 1, {"NAME"}},\

{"tokenBODY", 1, {"BODY"}},\

{"tokenDATA", 1, {"DATA"}},\

{"tokenEND", 1, {"END"}},\

{"tokenSEMICOLON", 1, {";"}},\

\

{"value", 1, {"value\_terminal"}},\

\

{"ident", 1, {"ident\_terminal"}},\

\

{"", 2, {"",""}}\

\

},\

176,\

"program"

///////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////

//#define DEFAULT\_MODE (DEBUG\_MODE | LEXICAL\_ANALISIS\_MODE)

#define DEFAULT\_MODE (DEBUG\_MODE | LEXICAL\_ANALYZE\_MODE | SYNTAX\_ANALYZE\_MODE | SEMANTIX\_ANALYZE\_MODE | MAKE\_ASSEMBLY | MAKE\_BINARY)  
  
def.h  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: def.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SUCCESS\_STATE 0

#define LEXICAL\_ANALYZE\_MODE 1 // lexicalAnalyze

#define MAKE\_LEXEMES\_SEQUENSE 2 // ADD MODE

#define SYNTAX\_ANALYZE\_MODE 4

#define MAKE\_AST 8 // ADD MODE

#define SEMANTIX\_ANALYZE\_MODE 16 // ADD MODE

#define MAKE\_PREPARE 32 // ADD MODE

#define MAKE\_C 64 // ADD MODE

#define MAKE\_ASSEMBLY 128 // ADD MODE

#define MAKE\_OBJECT 256 // ADD MODE

#define MAKE\_BINARY 512 // ADD MODE

#define RUN\_BINARY 1024 // ADD MODE

#define UNDEFINED\_MODE 16384

#define INTERACTIVE\_MODE 32768

#define SKIP\_INTERACTIVE\_IN\_INTERACTIVE\_MODE 65536

#define FULL\_COMPILER\_MODE 2048 // ?

#define DEBUG\_MODE 4096

//#define DECLENUM(NAME, ...) typedef enum {\_\_VA\_ARGS\_\_, size##NAME} NAME;

#define DECLENUM(NAME, ...) enum NAME {\_\_VA\_ARGS\_\_, size##NAME};

#define GET\_ENUM\_SIZE(NAME) size##NAME

#define SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(ARRAY, NAME)\

ARRAY[MULTI\_TOKEN\_##NAME][0] = (char\*)T\_##NAME##\_0;\

ARRAY[MULTI\_TOKEN\_##NAME][1] = (char\*)T\_##NAME##\_1;\

ARRAY[MULTI\_TOKEN\_##NAME][2] = (char\*)T\_##NAME##\_2;\

ARRAY[MULTI\_TOKEN\_##NAME][3] = (char\*)T\_##NAME##\_3;

//#define EXPAND\_MACRO(...) \_\_VA\_ARGS\_\_ // Проміжний макрос для розгортання

//

//#define SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY\_(ARRAY, QUADRUPLE\_ELEMENT\_INDEX, ...)\

//SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY(ARRAY, QUADRUPLE\_ELEMENT\_INDEX, EXPAND\_MACRO(\_\_VA\_ARGS\_\_) )

//

//#define TOKENS\_FOR\_MULTI\_TOKEN\_BITWISE\_NOT "~", "", "", ""

//

//SET\_QUADRUPLE\_STR\_MACRO\_IN\_ARRAY\_(tokenStruct, MULTI\_TOKEN\_BITWISE\_NOT, TOKENS\_FOR\_MULTI\_TOKEN\_BITWISE\_NOT)

//#define MAX\_TEXT\_SIZE 8192

//#define MAX\_WORD\_COUNT (MAX\_TEXT\_SIZE / 5)

//#define MAX\_LEXEM\_SIZE 1024

//#define MAX\_VARIABLES\_COUNT 256

//#define MAX\_KEYWORD\_COUNT 64

//

//#define KEYWORD\_LEXEME\_TYPE 1

//#define IDENTIFIER\_LEXEME\_TYPE 2 // #define LABEL\_LEXEME\_TYPE 8

//#define VALUE\_LEXEME\_TYPE 4

//#define UNEXPEXTED\_LEXEME\_TYPE 127

//

//#define LEXICAL\_ANALISIS\_MODE 1

//#define SEMANTIC\_ANALISIS\_MODE 2

//#define FULL\_COMPILER\_MODE 4

//

//#define DEBUG\_MODE 512

//

//+!///#define MAX\_PARAMETERS\_SIZE 4096

//+!///#define PARAMETERS\_COUNT 4

//+!///#define INPUT\_FILENAME\_PARAMETER 0

//

//#define DEFAULT\_MODE (LEXICAL\_ANALISIS\_MODE | DEBUG\_MODE)  
  
  
div.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: div.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define DIV\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeDivCode(B, C, M);

unsigned char\* makeDivCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
else.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: else.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define ELSE\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeElseCode(B, C, M);\

if (A ==\* B) C = makeSemicolonAfterElseCode(B, C, M);

unsigned char\* makeElseCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeSemicolonAfterElseCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
equal.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: equal.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define EQUAL\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeIsEqualCode(B, C, M);

unsigned char\* makeIsEqualCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
for.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: for.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define FOR\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeForCycleCode(B, C, M);\

if (A ==\* B) C = makeToOrDowntoCycleCode(B, C, M);\

if (A ==\* B) C = makeDoCycleCode(B, C, M);\

if (A ==\* B) C = makeSemicolonAfterForCycleCode(B, C, M);

unsigned char\* makeForCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeToOrDowntoCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeDoCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeSemicolonAfterForCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
generator.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: generator.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/config.h"

// TODO: CHANGE BY fRESET() TO END

#define C\_CODER\_MODE 0x01

#define ASSEMBLY\_X86\_WIN32\_CODER\_MODE 0x02

#define MACHINE\_X86\_WIN32\_CODER\_MODE 0x04

extern unsigned char generatorMode;

#define CODEGEN\_DATA\_TYPE int

#define START\_DATA\_OFFSET 512

#define OUT\_DATA\_OFFSET (START\_DATA\_OFFSET + 512)

#define M1 1024

#define M2 1024

//unsigned long long int dataOffsetMinusCodeOffset = 0x00003000;

#define dataOffsetMinusCodeOffset 0x00004000ull

//unsigned long long int codeOffset = 0x000004AF;

//unsigned long long int baseOperationOffset = codeOffset + 49;// 0x00000031;

#define baseOperationObjectOffset 0x0000018Bull

#define baseOperationOffset 0x000004AFull

#define putProcOffset 0x0000001Bull

#define getProcOffset 0x00000044ull

//unsigned long long int startCodeSize = 64 - 14; // 50 // -1

unsigned char detectMultiToken(struct LexemInfo\* lexemInfoTable, enum TokenStructName tokenStructName);

unsigned char createMultiToken(struct LexemInfo\*\* lexemInfoTable, enum TokenStructName tokenStructName);

#define MAX\_ACCESSORY\_STACK\_SIZE 128

extern struct NonContainedLexemInfo lexemInfoTransformationTempStack[MAX\_ACCESSORY\_STACK\_SIZE];

extern unsigned long long int lexemInfoTransformationTempStackSize;

unsigned char\* outBytes2Code(unsigned char\* currBytePtr, unsigned char\* fragmentFirstBytePtr, unsigned long long int bytesCout);

#if 1

unsigned char\* getObjectCodeBytePtr(unsigned char\* baseBytePtr, unsigned char generatorMode);

unsigned char\* getImageCodeBytePtr(unsigned char\* baseBytePtr, unsigned char generatorMode);

unsigned char\* makeCode(struct LexemInfo\*\* lastLexemInfoInTable/\*TODO:...\*/, unsigned char\* currBytePtr, unsigned char generatorMode);

void viewCode(unsigned char\* outCodePtr, unsigned long long int outCodePrintSize, unsigned char align);

#endif

unsigned long long int buildTemplateForCodeObject(unsigned char\* byteImage);

unsigned long long int buildTemplateForCodeImage(unsigned char\* byteImage);

void writeBytesToFile(const char\* output\_file, unsigned char\* byteImage, unsigned long long int imageSize);  
  
  
goto\_label.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: goto\_lable.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string>

#include <map>

//#include <utility>

#include <stack>

extern std::map<std::string, std::pair<unsigned long long int, std::stack<unsigned long long int>>> labelInfoTable;

#define LABEL\_GOTO\_LABELE\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeLabelCode(B, C, M);\

if (A ==\* B) C = makeGotoLabelCode(B, C, M);

unsigned char\* makeLabelCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeGotoLabelCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
greater.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: greater.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define GREATER\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeIsGreaterCode(B, C, M);

unsigned char\* makeIsGreaterCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
if\_then.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: if\_then.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define IF\_THEN\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeIfCode(B, C, M);\

if (A ==\* B) C = makeThenCode(B, C, M);\

if (A ==\* B) C = makeSemicolonAfterThenCode(B, C, M);

unsigned char\* makeIfCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeThenCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeSemicolonAfterThenCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
  
input.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: input.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define INPUT\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeGetCode(B, C, M);

unsigned char\* makeGetCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
less.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: less.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define LESS\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeIsLessCode(B, C, M);

unsigned char\* makeIsLessCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
lexica.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: lexica.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/////#define IDENTIFIER\_LEXEME\_TYPE 2

/////#define VALUE\_LEXEME\_TYPE 4

#define VALUE\_SIZE 4

#define MAX\_TEXT\_SIZE 8192

#define MAX\_WORD\_COUNT (MAX\_TEXT\_SIZE / 5)

#define MAX\_LEXEM\_SIZE 1024

#define MAX\_VARIABLES\_COUNT 256

#define MAX\_KEYWORD\_COUNT 64

#define KEYWORD\_LEXEME\_TYPE 1

#define IDENTIFIER\_LEXEME\_TYPE 2 // #define LABEL\_LEXEME\_TYPE 8

#define VALUE\_LEXEME\_TYPE 4

#define UNEXPEXTED\_LEXEME\_TYPE 127

#ifndef LEXEM\_INFO\_

#define LEXEM\_INFO\_

struct NonContainedLexemInfo;

struct LexemInfo {public:

char lexemStr[MAX\_LEXEM\_SIZE];

unsigned long long int lexemId;

unsigned long long int tokenType;

unsigned long long int ifvalue;

unsigned long long int row;

unsigned long long int col;

// TODO: ...

LexemInfo();

LexemInfo(const char\* lexemStr, unsigned long long int lexemId, unsigned long long int tokenType, unsigned long long int ifvalue, unsigned long long int row, unsigned long long int col);

LexemInfo(const NonContainedLexemInfo& nonContainedLexemInfo);

};

#endif

#ifndef NON\_CONTAINED\_LEXEM\_INFO\_

#define NON\_CONTAINED\_LEXEM\_INFO\_

struct LexemInfo;

struct NonContainedLexemInfo {

//char lexemStr[MAX\_LEXEM\_SIZE];

char\* lexemStr;

unsigned long long int lexemId;

unsigned long long int tokenType;

unsigned long long int ifvalue;

unsigned long long int row;

unsigned long long int col;

// TODO: ...

NonContainedLexemInfo();

NonContainedLexemInfo(const LexemInfo& lexemInfo);

};

#endif

extern struct LexemInfo lexemesInfoTable[MAX\_WORD\_COUNT];

extern struct LexemInfo\* lastLexemInfoInTable;

extern char identifierIdsTable[MAX\_WORD\_COUNT][MAX\_LEXEM\_SIZE];

void printLexemes(struct LexemInfo\* lexemInfoTable, char printBadLexeme/\* = 0\*/);

void printLexemesToFile(struct LexemInfo\* lexemInfoTable, char printBadLexeme, const char\* filename);

unsigned int getIdentifierId(char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\* str);

unsigned int tryToGetIdentifier(struct LexemInfo\* lexemInfoInTable, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE]);

unsigned int tryToGetUnsignedValue(struct LexemInfo\* lexemInfoInTable);

int commentRemover(char\* text, const char\* openStrSpc/\* = "//"\*/, const char\* closeStrSpc/\* = "\n"\*/);

void prepareKeyWordIdGetter(char\* keywords\_, char\* keywords\_re);

unsigned int getKeyWordId(char\* keywords\_, char\* lexemStr, unsigned int baseId);

char tryToGetKeyWord(struct LexemInfo\* lexemInfoInTable);

void setPositions(const char\* text, struct LexemInfo\* lexemInfoTable);

struct LexemInfo lexicalAnalyze(struct LexemInfo\* lexemInfoInPtr, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE]);

struct LexemInfo tokenize(char\* text, struct LexemInfo\*\* lastLexemInfoInTable, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], struct LexemInfo(\*lexicalAnalyzeFunctionPtr)(struct LexemInfo\*, char(\*)[MAX\_LEXEM\_SIZE]));  
  
  
lrbind.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: lrbind.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define LRBIND\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeLeftToRightBindCode(B, C, M);

unsigned char\* makeLeftToRightBindCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
mod.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: mod.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MOD\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeModCode(B, C, M);

unsigned char\* makeModCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
  
mul.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: mul.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MUL\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeMulCode(B, C, M);

unsigned char\* makeMulCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
not.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: not.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define NOT\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeNotCode(B, C, M);

unsigned char\* makeNotCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
nor\_equal.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: not\_equal.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define NOT\_EQUAL\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeIsNotEqualCode(B, C, M);

unsigned char\* makeIsNotEqualCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
  
null\_statement.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: null\_statement.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define NON\_CONTEXT\_NULL\_STATEMENT(A, B, C, M, R)\

if (A ==\* B) C = makeNullStatementAfterNonContextCode(B, C, M);

unsigned char\* makeNullStatementAfterNonContextCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
  
operand.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: operand.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define OPERAND\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeValueCode(B, C, M);\

if (A ==\* B) C = makeIdentifierCode(B, C, M);

unsigned char\* makeValueCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeIdentifierCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
or.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: or.hxx \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define OR\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeOrCode(B, C, M);

unsigned char\* makeOrCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
output.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: output.hxx \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define OUTPUT\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makePutCode(B, C, M);

unsigned char\* makePutCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
  
preparer.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: preparer.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int precedenceLevel(char\* lexemStr);

bool isLeftAssociative(char\* lexemStr);

bool isSplittingOperator(char\* lexemStr);

void makePrepare4IdentifierOrValue(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable);

void makePrepare4Operators(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable);

void makePrepare4LeftParenthesis(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable);

void makePrepare4RightParenthesis(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable);

unsigned int makePrepareEnder(struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable);

long long int getPrevNonParenthesesIndex(struct LexemInfo\* lexemInfoInTable, unsigned long long currIndex);

long long int getEndOfNewPrevExpressioIndex(struct LexemInfo\* lexemInfoInTable, unsigned long long currIndex);

unsigned long long int getNextEndOfExpressionIndex(struct LexemInfo\* lexemInfoInTable, unsigned long long prevEndOfExpressionIndex);

void makePrepare(struct LexemInfo\* lexemInfoInTable, struct LexemInfo\*\* lastLexemInfoInTable, struct LexemInfo\*\* lastTempLexemInfoInTable);  
  
  
  
repeat\_until.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: repeat\_until.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define REPEAT\_UNTIL\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeRepeatCycleCode(B, C, M);\

if (A ==\* B) C = makeUntileCode(B, C, M);\

if (A ==\* B) C = makeNullStatementAfterUntilCycleCode(B, C, M);

unsigned char\* makeRepeatCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeUntileCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeNullStatementAfterUntilCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

rlbind.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: rlbind.hxx \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define RLBIND\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeRightToLeftBindCode(B, C, M);

unsigned char\* makeRightToLeftBindCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
  
semantix.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: semantix.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#define COLLISION\_II\_STATE 128

#define COLLISION\_LL\_STATE 129

#define COLLISION\_IL\_STATE 130

#define COLLISION\_I\_STATE 132

#define COLLISION\_L\_STATE 136

#define COLLISION\_IK\_STATE 144

#define UNINITIALIZED\_I\_STATE 160

#define NO\_IMPLEMENT\_CODE\_STATE 256

unsigned long long int getDataSectionLastLexemIndex(LexemInfo\* lexemInfoTable, Grammar\* grammar);

int checkingInternalCollisionInDeclarations(LexemInfo\* lexemInfoTable, Grammar\* grammar, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\*\* errorMessagesPtrToLastBytePtr);

int checkingVariableInitialization(LexemInfo\* lexemInfoTable, Grammar\* grammar, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\*\* errorMessagesPtrToLastBytePtr);

int checkingCollisionInDeclarationsByKeyWords(char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\*\* errorMessagesPtrToLastBytePtr);

int semantixAnalyze(LexemInfo\* lexemInfoTable, Grammar\* grammar, char(\*identifierIdsTable)[MAX\_LEXEM\_SIZE], char\* errorMessagesPtrToLastBytePtr);  
  
  
  
semicolon.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: semicolon.hxx \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define NON\_CONTEXT\_SEMICOLON\_CODER(A, B, C, M, R)\

/\* (1) Ignore phase\*/if (A ==\* B) C = makeSemicolonAfterNonContextCode(B, C, M);\

/\* (2) Ignore phase\*/if (A ==\* B) C = makeSemicolonIgnoreContextCode(B, C, M);

unsigned char\* makeSemicolonAfterNonContextCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeSemicolonIgnoreContextCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
  
sub.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: sub.hxx \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SUB\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeSubCode(B, C, M);

unsigned char\* makeSubCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);  
  
  
syntax.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: syntax.h \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "../../include/def.h"

#include "../../include/generator/generator.h"

#include "../../include/lexica/lexica.h"

#define SYNTAX\_ANALYZE\_BY\_CYK\_ALGORITHM 0

#define SYNTAX\_ANALYZE\_BY\_RECURSIVE\_DESCENT 1

#define DEFAULT\_SYNTAX\_ANAlYZE\_MODE SYNTAX\_ANALYZE\_BY\_CYK\_ALGORITHM

using namespace std;

#define MAX\_RULES 356

#define MAX\_TOKEN\_SIZE 128

#define MAX\_RTOKEN\_COUNT 2 // 3

typedef struct {

char lhs[MAX\_TOKEN\_SIZE];

int rhs\_count;

char rhs[MAX\_RTOKEN\_COUNT][MAX\_TOKEN\_SIZE];

} Rule;

typedef struct {

Rule rules[MAX\_RULES];

int rule\_count;

char start\_symbol[MAX\_TOKEN\_SIZE] ;

} Grammar;

extern Grammar grammar;

#define DEBUG\_STATES

bool recursiveDescentParserRuleWithDebug(const char\* ruleName, int& lexemIndex, LexemInfo\* lexemInfoTable, Grammar\* grammar, int depth, const struct LexemInfo\*\* unexpectedLexemfailedTerminal);

//bool cykAlgorithmImplementation(struct LexemInfo\* lexemInfoTable, Grammar\* grammar);

int syntaxAnalyze(LexemInfo\* lexemInfoTable, Grammar\* grammar, char syntaxlAnalyzeMode, char\* astFileName, char\* errorMessagesPtrToLastBytePtr);  
  
  
  
while.h  
#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* N.Kozak // Lviv'2024-2025 // cw\_sp2\_\_2024\_2025 \*

\* file: while.hxx \*

\* (draft!) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define WHILE\_CODER(A, B, C, M, R)\

if (A ==\* B) C = makeWhileCycleCode(B, C, M);\

if (A ==\* B) C = makeNullStatementWhileCycleCode(B, C, M);\

if (A ==\* B) C = makeContinueWhileCycleCode(B, C, M);\

if (A ==\* B) C = makeExitWhileCycleCode(B, C, M);\

if (A ==\* B) C = makeEndWhileAfterWhileCycleCode(B, C, M);

unsigned char\* makeWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeNullStatementWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeContinueWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeExitWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);

unsigned char\* makeEndWhileAfterWhileCycleCode(struct LexemInfo\*\* lastLexemInfoInTable, unsigned char\* currBytePtr, unsigned char generatorMode);