Національний університет “Львівська політехніка”

Інститут комп’ютерної техніки, автоматики та метрології

Кафедра СКС

****

**Курсова робота**

з предмету “Системне програмування”

На тему: “Розробка системних програмних модулів та компонент систем програмування”

Індивідуальне завдання: “Розробка транслятора з вхідної мови програмування”

Варіант № 6

Виконав: ст. групи КІ-308

Дідик О. М.

Прийняв:   
викладач каф. СКС  
Козак Н.Б.

Львів – 2024

**Анотація**

Курсова робота з дисципліни "Системне програмування" вміщає в собі весь матеріал, який ми вивчали протягом даного курсу. Вона узагальнює матеріал і закріпляє навички які ми набули при вивченні матеріалу.

В ній ми маємо продемонструвати розробку транслятора з вхідної мови програмування, яка нам була задана, на мову асемблер, компілювати код і створити виконавчий файл. Транслятор повинен виконувати: лексичний аналіз, синтаксичний аналіз, семантичний аналіз, виводити список помилок при наявності та попереджень.

В цій курсовій роботі буде використовуватися лексичний аналізатор на базі скінченного автомата та на основі магазинного автомата.

Зміст

[**Анотація** 2](#_Toc501563928)

[**Завдання на курсову роботу** 4](#_Toc501563929)

[**1. Огляд методів та способів проектування трансляторів** 6](#_Toc501563930)

[**2. Формальний опис вхідної мови програмування** 8](#_Toc501563931)

[**2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура** 8](#_Toc501563932)

[**2.2. Термінальні символи та ключові слова.** 9](#_Toc501563933)

**Завдання на курсову роботу**

Тема: Розробка транслятора з вхідної мови програмування.

- початок програми Program;

- початок блоку даних Var

- типи даних: Int\_4;

- оператор вводу: Scan;

- оператор виводу: Print;

- блок тіла програми: Program ; Start Var …; Finish

- оператори If-else (С) Goto (С) For-To(Паскаль) For-Downto(Паскаль) While (Бейсік) Repeat-Until (Паскаль);

- регістр ключових слів: Up-Low перший символ Up;

- регістр ідентифікаторів: Low-Up8 перший символ Low;

- операції арифметичні: ++, --, \*\*, Div,Mod;

- операції порівняння: Eg; Ne; Le; Ge

- операції логічні: Not; And; Or

- коментар: {\* ... \*}

- ідентифікатори змінних, числові константи, рядкові константи;

- оператор присвоєння: ==>

Для отримання виконавчого файлу з вихідного асемблерного коду потрібно використовувати masm32.exe.

**Вступ**

Компілятор (англ. Compiler від англ. to compile збирати в ціле) - комп'ютерна програма, що перетворює (компілює) програмний код, написаний певною мовою програмування, на семантично еквівалентний код в іншій мові програмування, який, як правило, необхідний для виконання програми машиною.

Транслятор – це той самий компілятор, з тею різницею, що генерує він не об’єктний код, а код на іншій мові програмування.

Процес компіляції як правило складається з декількох етапів: лексичного, синтаксичного та семантичного (типозалежного) аналізів, генерації проміжного коду, оптимізації та генерації результуючого машинного коду. Крім цього, програма як правило залежить від сервісів, наданих операційною системою і сторонніми бібліотеками (наприклад, файловий ввід-вивід або графічний інтерфейс), і машинний код програми необхідно пов'язати з цими сервісами. Для зв'язування зі статичними бібліотеками виконується редактор зв'язків або компонувальник, а з операційною системою і динамічними бібліотеками зв'язування виконується на початку виконання програми завантажувача.

Основні задачі, які виконуються різними компіляторами та трансляторами, по суті, одні і ті ж. Розуміючи ці задачі, існує можливість створювати транслятори для різних початкових мов і цільових машин з використанням одних і тих же базових технологій.

В даній курсовій роботі створюватимется транслятор мови програмування заданої варіантом, який включає:

- лексичний аналізатор, здатний розпізнавати лексеми, що є описані в формальному описі мови програмування.

- синтаксичний аналізатор на основі методу зсув-згортка.

- генератор коду, що генерує код який відповідає кожній конструкції вхідної мови.

Також повинне бути забеспечене виявлення лексичних та синтаксичних помилок, виконана загальна перевірка роботи компілятора.

1. ОГЛЯД МЕТОДІВ ТА СПОСОБІВ ПРОЄКТУВАННЯ ТРАНСЛЯТОРІВ

Створення компіляторів відбувається в певних конкретних умовах: для різних мов, для різних цільових платформ, з різними вимогами для створення компіляторів. Є такі методи створення компіляторів:

1. Прямий метод - цільовою мовою і мовою реалізації є асемблер.

2. Метод розкрути - саме цей метод і використовується у даній курсовій роботі, тобто вибирається інструмент (в даній курсовій це мова асемблер), для якого вже існує компілятор.

3.Використання крос-трансляторів.

4.З використанням віртуальних машин – дає спосіб отримати переносимо програму.

5. Компіляція на ходу.

В даній курсовій роботі згідно із завданням для непарних варіантів необхідно реалізувати висхідний метод граматичного розбору.

При такій стратегії дерево синтаксичного аналізу будується, рухаючись від листя (вхідної програми, яка розглядається як рядок символів) до кореня дерева (аксіоми граматики). Аналізатор (розпізнавач) шукає частину рядка, яку можна звести до нетермінального символу. Таку частину рядка називають фразою. У більшості висхідних розпізнавачів відшукується найлівіша фраза, що безпосередньо зводиться до нетермінального символу (така фраза називається основою). Основа заміняється нетермінальним символом. У отриманому рядку знову відшукується основа, заміняється нетермінальним символом і т.д.

У загальному випадку процедуру побудови висхiдного розпiзнавача за заданою граматикою можна описати в такий спосiб:

1. Визначити для даної граматики функцiї ВПЕРВ i ВПIСЛЯ.
2. Побудувати детермiновану таблицю переходiв, що має по одному стовпцю для кожного граматичного символу i по одному рядку для кожного граматичного входження i маркера дна.
3. Якщо таблиця, побудована на кроцi 2, виходить недетермiнованою (має бiльше одного стану), то потрiбно перетворити цю таблицю в детермiновану, розглядаючи її як недетермiновану таблицю переходiв кiнцевого автомата з початковим станом *h*0.
4. Стани, отриманi на кроцi 3 (крiм стану, що вiдповiдає порожнiй множинi), варто використовувати як магазиннi символи. Отримана таблиця переходiв може мiстити переходи в порожню множину. Такi елементи варто розумiти як забороненi i розглядати переходи в них як помилки.
5. Керуючу таблицю заповнюють рядок за рядком вiдповiдно до множини граматичних входжень, що позначають рядки.

Фази лексичного (ЛА) та синтаксичного (СА) аналізів розкладають початкову програму на частини. Генерація коду проміжною мовою (ГПК), оптимізація (ОК) та генерація коду (ГК) асемблера синтезують програму на вихідній мові. Керування таблицями (КТ) та обробка помилками (ОП) використовуються на всіх фазах трансляції.

Лексичний аналіз об`єднує літери в лексеми - службові слова, ідентифікатори, знаки операцій та пунктуації. Лексеми можна кодувати цілими числами, наприклад, do-одиницею, "+" - двійкою, ідентифікатор - трійкою, константу - четвіркою тощо. До коду лексем ідентифікаторів і констант додається ще одна величина - вид чи значення лексеми.

Синтаксичний аналіз групує лексеми в синтаксичні структури, які можуть бути складовими інших синтаксичних структур; наприклад, А+В може входити в оператор чи вираз. Як рекурсивні структури даних, вони зображуються (явно чи неявно) у формі дерева з лексемами в вузлах і з позначенням синтаксичних конструкцій у внутрішніх вузлах. Крони піддерев відтворюють частини програми відповідної синтаксичної конструкції.

1. ФОРМАЛЬНИЙ ОПИС ВХІДНОЇ МОВИ ПРОГРАМУВАННЯ
   1. **Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура**

Розширена форма Бекуса-Наура (скор. РБНФ) - формальна система визначення синтаксису, в якій одні синтаксичні категорії послідовно визначаються через інші. Використовується для опису контекстно-вільних формальних граматик. Запропоновано Никлаусом Віртом. Є розширеною переробкою форм Бекуса-Наура, відрізняється від БНФ більш «вмістимими» конструкціями, що дозволяють при тій же виразності дозволяє спростити і скоротити в обсязі опис.

labelpoint = label ,";";

gotolabel = "Goto",ident,";";

program\_name = ident;

value\_type = "Int\_4",ident, {"," , ident };

declaration\_ident = ident;

other\_declaration = ",",ident;

declaration = value\_type,decrlaration\_ident,{ other\_declaration\_ident } ;

operation\_not = "Not" , inseparable\_expression ;

and\_action = "And" , inseparable\_expression ;

or\_action = "Or" , high\_prioryty\_expression ;

equal\_action = "Eg" , middle\_prioryty\_expression ;

not\_equal\_action = "Ne" , middle\_prioryty\_expression ;

less\_or\_equal\_action = "Le" , middle\_prioryty\_expression ;

greater\_or\_equal\_action = "Ge" , middle\_prioryty\_expression ;

add\_action = "++" , high\_prioryty\_expression ;

sub\_action = "--" , high\_prioryty\_expression ;

mul\_action = "\*\*" , inseparable\_expression ;

div\_action = "Div" , inseparable\_expression ;

mod\_action = "Mod" , inseparable\_expression ;

unary\_operation = operation\_not ;

inseparable\_expression = group\_expression | unary\_operation | ident\_read | value\_read;

high\_prioryty\_left\_expression = group\_expression | unary\_operation | ident\_read | value\_read ;

high\_prioryty\_action = mul\_action | div\_action | mod\_action | and\_action ;

high\_prioryty\_expression = high\_prioryty\_left\_expression , { high\_prioryty\_action } ;

middle\_prioryty\_left\_expression = high\_prioryty\_expression | group\_expression | unary\_operation | ident\_read | value\_read ;

middle\_prioryty\_action = add\_action | sub\_action | or\_action;

middle\_prioryty\_expression = middle\_prioryty\_left\_expression , { middle\_prioryty\_action } ;

low\_prioryty\_left\_expression = middle\_prioryty\_expression | high\_prioryty\_expression | group\_expression | unary\_operation | ident\_read | value\_read ;

low\_prioryty\_action = less\_or\_equal\_action | greater\_or\_equal\_action | equal\_action | not\_equal\_action ;

low\_prioryty\_expression = low\_prioryty\_left\_expression , { low\_prioryty\_action } ;

group\_expression = "(" , low\_prioryty\_expression , ")" ;

bind = ident\_write, "==>" low\_prioryty\_expression ;

if\_expression = expression

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

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

cond\_block = tokenIF, "If"; if\_expression; body\_for\_true; [body\_for\_false]

cycle\_begin\_expression = low\_prioryty\_expression;

cycle\_counter = ident ;

cycle\_counter\_last\_value = value ;

cycle\_body = "Do" , statement , { statement } ;

forto\_cycle = "For" , cycle\_begin\_expression , "==>" , cycle\_counter , "To" , cycle\_counter\_last\_value , cycle\_body , ";" ;

while\_cycle\_head\_expression = low\_prioryty\_expression

while\_cycle = "While" , while\_cycle\_head\_expression , { statement } , ";" ;

do\_while\_cycle\_cond = low\_prioryty\_expression ;

do\_while\_cycle = "Do" , { statement } , "While" , do\_while\_cycle\_cond ;

tokenCONTINUE = "Continue" ;

tokenWHILE = "While" ;

tokenEXIT = "Exit" ;

continue\_while = tokenCONTINUE , tokenWHILE ;

exit\_while = tokenEXIT , tokenWHILE ;

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

input = "Scan" , "(" , ident\_write , ")" ;

output = "Print", "(", low\_prioryty\_expression, ")";

statement = recursive\_descent\_end\_point | bind | cond\_block | forto\_cycle | while\_cycle | do\_while\_cycle | labeled\_point | goto\_label | input | output;

program = program\_name , ";" , "Start" , "Var" , [ declaration ] , ";" , { statement } , "Finsih" ;

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 } | "0") ;

value = [ sign ] , unsigned\_value ;

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 = low\_letter, upper\_letter, upper\_letter, upper\_letter, upper\_letter, upper\_letter, upper\_letter, upper\_letter, upper\_letter;

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';

**2.2. Термінальні символи та ключові слова**

**Термінальні символи** — це всі символи, що з'являються в конструкціях мови та є «кінцевими» в граматичних правилах.

1. **Program** — початок програми, визначає назву програми.
2. **Start** — початок блоку програми.
3. **Var** — блок змінних.
4. **Finish** — кінець програми або циклу.
5. **Scan** — оператор введення даних.
6. **Print** — оператор виведення даних (змінні та рядкові константи).
7. **==>** — оператор присвоєння.
8. **For ... To ... Do** — оператор циклу з умовами.
9. **++** — операція додавання.
10. **--** — операція віднімання.
11. **\*\*** — операція множення.
12. **Div** — операція ділення.
13. **Mod** — операція знаходження залишку від ділення.
14. **Eg** — операція перевірки на рівність.
15. **Ne** — перевірка на нерівність.
16. **Le** — операція «менше або рівно».
17. **Ge** — операція «більше або рівно».
18. **Not** — операція логічного заперечення.
19. **And** — логічна кон'юнкція (AND).
20. **Or** — логічна диз'юнкція (OR).
21. **Int\_4** — 32-бітові цілі числа.
22. **Boolean** — логічні змінні (істинно/хибно).
23. **True** та **False** — логічні константи.
24. **{\*...\*}** — коментар.
25. **"..."** — рядкова константа.
26. **;** — кінець оператора.
27. **(** та **)** — дужки для виразу.
28. **,** — розділювач між деклараціями змінних.

Крім того, в мові використовуються:

* Латинські літери (a-z, A-Z).
* Арабські цифри (0-9).
* Символи табуляції, пробіли та символи переходу на нову стрічку.
  1. **Перевірка роботи транслятора за допомогою тестових задач.**

4.3.1. 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. Вивести на екран:

А + В (результат операції додавання);

А - В (результат операції віднімання);

А \* В (результат операції множення);

А / В (результат операції ділення);

А % В (результат операції отримання залишку від ділення).

3. Обрахувати значення виразів

Х = (А - В) \* 10 + (А + В) / 10

У = Х + Х % 10

4. Вивести значення Х і У на екран.

Напишемо програму на вхідній мові програмування:

Program fIRSTASKK;

Var

Int\_4 nUMBERAAA, nUMNERBBB, rRESULTXX, rRESULTYY;

Start

Scan nUMBERAAA;

Scan nUMNERBBB;

Print nUMBERAAA + nUMNERBBB;

Print nUMBERAAA - nUMNERBBB;

Print nUMBERAAA Mul nUMNERBBB;

If nUMNERBBB Ne 0 Then

Put nUMBERAAA Div nUMNERBBB;

Else

Print 'Division by zero!';

If nUMNERBBB Ne 0 Then

Print nUMBERAAA Mod nUMNERBBB;

Else

Print 'Modulo by zero!';

(nUMBERAAA -- nUMNERBBB) Mul 10 ++ (nUMBERAAA ++ nUMNERBBB) Div 10 ==> rRESULTXX;

rRESULTXX ++ rRESULTXX Mod 10 ==> rRESULTYY;

Print rRESULTXX;

Print rRESULTYY;

Finish

4.3.2. 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:

2. Знайти найбільше з них і вивести його на екран. Використання простого умовного оператора:

3. Вивести на екран число 1, якщо усі числа однакові (логічний вираз в умовному операторі має виглядати так: «(А=В) і (А=С) і (В=С)»), інакше вивести 0.

4. Вивести на екран число -1, якщо хоча б одне з чисел від’ємне (логічний вираз в умовному операторі має виглядати так: «(А<0) або (В<0) або (С<0)»), інакше вивести 0.

5. Вивести на екран число 10, якщо число А більше за суму чисел В і С (логічний вираз в умовному операторі має виглядати так: «!(А<(В+С))»), інакше вивести 0.

Program mAIBRANCH;

Var

Int\_4 nUMBERAA, nUMBERBB, nUMBERCC, mMAXNAMIM;

Start

Scan nUMBERAAA;

Scan nUMBERBBB;

Scan nUMBERCCC;

If nUMBERAAA Ge nUMBERBBB Then

If nUMBERAAA Ge nUMBERCCC Then

nUMBERAAA ==> mMAXNAMIM

Else

nUMBERCCC ==> MMAXNAMIM

Else

If nUMBERBBB Ge nUMBERCCC Then

nUMBERBBB ==> mMAXNAMIM

Else

nUMBERCCC ==> mMAXNAMIM;

Put MMAXNAMIM;

If (nUMBERAAA Eg nUMBERBBB) And (nUMBERAAA Eg nUMBERCCC) And (nUMBERBBB Eg nUMBERCCC) Then

Print 1

Else

Print 0;

If (nUMBERAA Le 0) Or (nUMBERBB Le 0) Or (nUMBERCC Le 0) Then

Print -1

Else

Print 0;

If Not( nUMBERAAA Le (nUMBERBBB ++ nUMBERCCC)) Then

Print 10

Else

Print 0;

Finish

4.3.3 Тестова програма «Циклічний алгоритм»

1. Ввести два числа А і В, причому А<В (імена змінних можуть бути іншими і мають відповідати правилам запису ідентифікаторів згідно індивідуального завдання). Використання простого оператора циклу:

2. Вивести на екран квадрати чисел від А до В включно. Використання вкладеного оператора циклу:

3. Обрахувати Х=А\*В за наступним алгоритмом: Х = 0 Цикл від 1 до А з кроком 1 Цикл від 1 до В з кроком 1 Х = Х + 1

4. Вивести значення Х на екран.

Program mAILOOPAL;

Var

Int\_4 nUMBERAAA, nUMBERBBB, rRESULTXX;

Int\_4 IIIIIIIII, JJJJJJJJJ;

Start

Scan nUMBERAAA;

Scan nUMBERBBB;

If nUMBERAAA Le nUMBERBBB Then

Start

For nUMBERAAA ==> IIIIIIII To NUMBERBB Do

Print IIIIIIII Mul IIIIIIII;

rRESULTXX ==> 0;

For 1 ==> IIIIIIII To nUMBERAAA Do

For 1 ==> JJJJJJJJ To nUMBERBBB Do

rRESULTXX + 1 ==> rRESULTXX;

Print rRESULTXX;

Finish

Else

Print 'nUMBERAAA must be less than nUMBERBBB!';

Finish