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 № 22**

Виконала: ст. групи КІ-308  
Чорненька С.Ч.  
  
Прийняв: викладач каф. СКС  
Козак Н. Б.

**Львів-2024**

Зміст

[Анотація 3](#_Toc185360804)

[Завдання на курсову роботу: 4](#_Toc185360805)

[Вступ 5](#_Toc185360806)

[1. Огляд методів та способів проектування трансляторів 6](#_Toc185360807)

[2. Формальний опис вхідної мови програмування 8](#_Toc185360808)

[2.1. Деталізований опис вхідної мови в термінах розширеної нотації Бекуса-Наура 8](#_Toc185360809)

[2.2. Термінальні символи та ключові слова. 10](#_Toc185360810)

[3. Перевірка роботи транслятора за допомогою тестових задач. 11](#_Toc185360811)

# **Анотація**

Курсова робота з дисципліни "Системне програмування" вміщує в собі весь матеріал, вивчений протягом даного курсу а також задією отримані навички.

В даній роботі повинно бути продемонстровано розробку транслятора з вхідної мови програмування, яка була задана згідно з варіантом, на асемблерну мову, а саме створити виконуваний файл. Транслятор повинен виконувати: лексичний аналіз, синтаксичний аналіз, семантичний аналіз.

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

# **Завдання на курсову роботу:**

- початок програми mainprogram

- початок блоку даних start data

- типи даних: int32\_t

- оператор вводу: scan

- оператор виводу: print

- блок тіла програми: end

- оператор присвоєння: ==>

- оператор:

if-else (C)

goto (C)

for-to (Паскаль)

for-downto (Паскаль)

while (Бейсік)

repeat-until (Паскаль)

- регістр ключових слів: Low

- регістр ідентифікаторів: Low-Up8

- операції арифметичні: add; sub;\*; /; %

- операції порівняння: eg; ne ; >>; <<

- операції логічні:not; and; or

- коментар: {\*….\*}

.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.

# **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 тією 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:

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

# **1. Огляд методів та способів проектування 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.

При методі граматичного розбору, 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.

# **2. Формальний опис вхідної мови програмування**

# **2.1. 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 = "int32\_t",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 = "<<" , middle\_prioryty\_expression ;

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

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

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

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

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

mod\_action = "%" , 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\_priority\_left\_expression = middle\_priority\_expression | high\_priority\_expression | group\_expression | unary\_operation | ident\_read | value\_read ;

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

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

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

bind = ident\_write , "==>", low\_priority\_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\_priority\_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 } , ";" ;

tokenCONTINUE = "continue" ;

tokenWHILE = "while" ;

tokenEXIT = "exit" ;

continue\_while = tokenCONTINUE , tokenWHILE ;

exit\_while = tokenEXIT , tokenWHILE ;

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

repeat\_until\_cycle\_cond = low\_prioryty\_expression ;

repeat\_until\_cycle = "repeat" , { statement } , "until" , repeat\_until\_cycle\_cond ;

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

output = "print", "(", low\_priority\_expression, ")";

statement = recursive\_descent\_end\_point | bind | if\_block | for\_downto\_cycle | while\_cycle | do\_while\_cycle | label\_point | goto\_label | input | output;

program = "mainprogram" , ";" , "start data" , { statement } , "end" ;

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 = letter\_in\_lower\_case, letter\_in\_upper\_case , letter\_in\_upper\_case ,letter\_in\_upper\_case ,letter\_in\_upper\_casee ,letter\_in\_upper\_case , letter\_in\_upper\_case, letter\_in\_upper\_case, letter\_in\_upper\_case

# **2.2. Термінальні символи та ключові слова.**

Якщо ми маємо якусь мову задану [граматикою](http://cybportal.univ.kiev.ua/wiki/%D0%93%D1%80%D0%B0%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0" \o "Граматика), то термінальними символами є всі символи, які з'являються в конструкціях мови. Термінальний означає кінцевий.

“ mainprogram ” – початок програми  
“data” – блок даних.  
“ start ” – початок блоку коду.  
“ end ” – кінець тіла програми (циклу).  
“scan” – оператор вводу змінних.  
“print” – оператор виводу (змінних і рядкових констант).  
“==>” - оператор присвоєння.

“” - пуста строка

“;”-кінець рядка програми  
“if-else” – умовний оператор.   
“goto” – оператор безумовного переходу.  
“for-to” – оператор циклу з нарощуванням.  
“for-downto” – оператор циклу зі зменшенням.  
“while” – оператор циклу з умовою на початку.  
“repeat-until “– оператор циклу з умовою на кінці.  
“add” - операція додавання.  
“sub” – операція відімання.  
“\*” – операція множення.  
“/” – операція ділення.

“%” – операція знаходження залишку від ділення.  
“eg” – рівність.   
“ne” – нерівність.  
“>>” - більше.  
“<<” - менше.

“not” – операція логічного заперечення.  
“and” – кон’юнкція.  
“or” – диз’юнкція.  
“int32\_tt” – Тип даних для 16-розрядних цілих.  
“{\*\*}”... – коментар.  
““” – початок/завершення рядкової константи при операції виводу;

“,” – розділювач між деклараціями змінних;

“(“– відкриваюча дужка;

“)” – закриваюча дужка;

Як термінальні символи використовуються також усі арабські цифри (0–9), латинські букви (a-z, A-Z), символи табуляції, символ переходу на нову стрічку, пробіл.

# **3. 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. 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:

mainprogram;

start data

Int32\_tt nUMAAAAAA, nUMBBBBBB, rESXXXXXX, rESYYYYYY;

scan nUMAAAAAA;

scan nUMBBBBBB;

print nUMAAAAAA add nUMBBBBBB;

print nUMAAAAAA sub nUMBBBBBB;

print nUMAAAAAA \* nUMBBBBBB;

if NUMBBB ne 0 then

print nUMAAAAAA / nUMBBBBBB;

else

print 'Division by zero!';

if nUMBBBBBB ne 0 then

print nUMAAAAAA % nUMBBBBBB;

else

print 'Modulo by zero!';

(nUMAAAAAA sub nUMBBBBBB) % 10 add (nUMAAAAAAadd nUMBBBBBB) / 10 ==> rESXXXXXX;

rESXXXXXXadd rESXXXXXX% 10 ==> rESYYYYYY;

print rESXXXXXX;

print rESYYYYYY;

end

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.

maintprogram;

start data

int32\_t nUMAAAAAA, nUMBBBBBB, nUMCCCCCC, mAXNUMMMM;

scan nUMAAAAAA;

scan nUMBBBBBB;

scan nUMCCCCCC;

if nUMAAAAAA >> nUMBBBBBB then

if nUMAAAAAA >> nUMCCCCCC then

nUMAAAAAA ==> mAXNUMMMM

else

nUMCCCCCC==> mAXNUMMMM

else

if nUMBBBBBB >> nUMCCCCCC then

nUMBBBBBB ==> mAXNUMMMM

else

nUMCCCCCC ==> mAXNUMMMM;

print mAXNUMMMM;

if (nUMAAAAAA eg nUMBBBBBB) and (nUMAAAAAA eg nUMCCCCCC) and (nUMBBBBBB eg nUMCCCCCC) then

print 1

else

print 0;

if (nUMAAAAAA << 0) or (nUMBBBBBB<< 0) or (nUMCCCCCC<< 0) then

print 0 sub 1

else

print 0;

if ( nUMAAAAAA << (nUMBBBBBB add nUMCCCCCC)) then

print 10

else

print 0;

end

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.

mainprogram;

start data

int32\_t nUMAAAAAA , nUMBBBBBB , rESXXXXXX;

int32\_t iIIIIIIII, jJJJJJJJJ;

scan nUMAAAAAA ;

scan nUMBBBBBB ;

if nUMAAAAAA << nUMBBBBBB then

start

for nUMAAAAAA ==> iIIIIIIII to nUMBBBBBB do

print iIIIIIIII\* iIIIIIIII;

rESXXXXXX ==> 0;

for 1 ==> jJJJJJJJJ to nUMAAAAAA do

for 1 ==> jJJJJJJJJ to nUMBBBBBB do

rESXXXXXX add 1 ==> rESXXXXXX;

print rESXXXXXX;

end

else

print ' nUMAAAAAA must be less than nUMBBBBBB !';

end