Міністерство освіти і науки України

Національний університет "Львівська Політехніка"

Кафедра ЕОМ



**Пояснювальна записка**

до курсового проєкту "Системне програмування"

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

Індивідуальне завдання

“Розробка транслятора з вхідної мови програмування”

Варіант 7

Виконав студент групи КІ-308:

Коноваленко В.Р.

Перевірив:

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 виконуваного файлу на виході розробленого транслятора скористатися середовищем Microsoft Visual Studio або будь-яким іншим.
3. Мова розробки транслятора: C/C++.
4. Реалізувати графічну 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 С або асемблера;*

*об’єктний файл;*

*виконуваний файл.*

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

**Деталізація завдання на проектування:**

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

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

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

**Деталізований опис власної мови програмування:**

Анотація

В даній курсовій роботі розроблено нову мову програмування, яка спрямована на спрощення процесу розробки алгоритмів. Основною метою проєкту є створення чіткої та зрозумілої граматики, яка відповідає сучасним вимогам програмування, що дозволяє користувачам з легкістю реалізовувати обчислювальні задачі.

В ній маємо продемонструвати розробку транслятора з вихідної мови програмування, яка була задана, компілювати товар і створити виконавчий код.

Зміст

Вступ 6   
1. Огляд методів та способів проектування трансляторів 7  
2. Формальний опис вхідної мови програмування 8   
2.1 Деталізований опис вхідної мови в термінах розширеної нотації Бекуса – Наура 8  
2.2 Термінальні символи та ключові слова 11

Вступ

**Компілятор** — це спеціальна програма, яка перетворює вихідний код, написаний мовою програмування високого рівня (такою як C, C++, Java, Python тощо), в об'єктний код або машинний код, який може виконуватися комп'ютером.

**Транслятор** — це програма, яка виконує перетворення коду з однієї мови програмування в іншу. У ширшому сенсі транслятор може включати компілятори, інтерпретатори та інші засоби для перетворення або аналізу програмного коду.

Сучасне програмування зазнає постійних змін, пов’язаних із розвитком технологій, нових парадигм і методів розробки програмного забезпечення. У зв’язку з цим виникає необхідність у створенні нових мов програмування, які б відповідали вимогам сучасних задач та дозволяли б ефективно реалізовувати алгоритми різного рівня складності. Розробка власної мови програмування є актуальною проблемою, оскільки наявні мови не завжди відповідають специфічним потребам розробників, не забезпечують достатньої гнучкості або не є достатньо простими для освоєння.

Актуальність даної курсової роботи полягає у створенні мови програмування, яка поєднує в собі простоту синтаксису з потужними можливостями для розв’язання різноманітних задач. Сьогодні програмісти часто стикаються з проблемами, пов’язаними з високим рівнем складності і громіздкістю синтаксису наявних мов, що призводить до зниження продуктивності праці та ускладнює навчання нових користувачів. Таким чином, існує потреба в мові, що надає зрозумілі інструменти для опису алгоритмів, забезпечує легкість вчитування коду та знижує поріг входження для новачків.

1. Огляд методів та способів проєктування трансляторів

Проєктування трансляторів є важливим етапом у розробці мов програмування, оскільки саме транслятори забезпечують перетворення вихідного коду на машинний код, що може виконуватись комп'ютером. Існує кілька методів і способів, які використовуються у цій галузі.

**1. Методика побудови трансляторів**

Транслятори можуть бути реалізовані за допомогою різних методик. Найпоширенішими є:

* **Процесори**: це транслятори, які обробляють вихідний код поетапно, зазвичай, у декілька проходів. Кожен прохід відповідає за певний етап компіляції, наприклад, лексичний, синтаксичний, семантичний аналіз.
* **Однопрохідні транслятори**: ці транслятори виконують усі етапи компіляції в одному проході. Це може бути ефективним для простих мов програмування, але обмежує можливості аналізу.

**2. Структурні методи**

Для побудови трансляторів використовуються різні структурні методи:

* **Деревоподібні структури**: представляють вихідний код у вигляді дерева, що відображає синтаксичну структуру програми. Це дозволяє легко виконувати семантичний аналіз та оптимізацію.
* **Регулярні вирази**: використовуються для лексичного аналізу. Вони дозволяють описувати шаблони токенів, які будуть зчитуватись транслятором.

**3. Методи оптимізації**

Оптимізація коду є важливим аспектом роботи трансляторів:

* **Оптимізація на етапі синтаксичного аналізу**: передбачає виявлення та усунення надлишкових конструкцій під час побудови дерева синтаксичного аналізу.
* **Оптимізація на етапі генерації коду**: включає перетворення середнього коду в оптимізований машинний код. Цей процес може включати видалення мертвого коду, фузію циклів та інші техніки.

**4. Інструменти для розробки трансляторів**

Сьогодні існує багато інструментів, які спрощують процес проєктування трансляторів:

* **Лексери та парсери**: такі як Flex та Bison, які автоматизують створення лексичних і синтаксичних аналізаторів на основі визначеної граматики.
* **IDE та середовища розробки**: інструменти, які забезпечують зручний інтерфейс для розробки трансляторів, включаючи налагодження та тестування.

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.

**Опис вхідної мови програмування у термінах розширеної форми Бекуса-Наура:**

<програма> ::= program <назва> start var <блок\_змінних> <тіло\_програми> finish.

<блок\_змінних> ::= <оголошення> {; <оголошення>}.

<оголошення> ::= <тип> <змінна> {, <змінна>}.

<тип> ::= int16\_t.

<змінна> ::= <ідентифікатор> [-> <константа>].

<ідентифікатор> ::= <літера> {<літера\_або\_цифра>}.

<назва> ::= <літера> {<літера\_або\_цифра>}.

<літера\_або\_цифра> ::= <літера> | <цифра>.

<літера> ::= 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 |

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.

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

<константа> ::= [-]<цифра> {<цифра>}.

<тіло\_програми> ::= <оператор> {<оператор>}.

<оператор> ::= <присвоєння> | <ввід> | <вивід> | <умова> | <складений\_оператор> | <цикл>.

<присвоєння> ::= <ідентифікатор> ==> <арифметичний\_вираз> ;.

<арифметичний\_вираз> ::= <термін> {+ <термін> | - <термін>}.

<термін> ::= <множник> { \* <множник> | / <множник> | % <множник>}.

<множник> ::= <ідентифікатор> | <константа> | (<арифметичний\_вираз>).

<умова> ::= if <логічний\_вираз> then <оператор> [else <оператор>].

<логічний\_вираз> ::= <порівняння> {or <порівняння>}.

<порівняння> ::= <арифметичний\_вираз> <оператор\_порівняння> <арифметичний\_вираз> |

!(<логічний\_вираз>) | (<логічний\_вираз>).

<оператор\_порівняння> ::= eg | ne | >= | <=.

<логічний оператор> ::= !| and| or.

<складений\_оператор> ::= start <тіло\_програми> finish.

<умова> ::= if <логічний\_вираз> then <оператор> [else <оператор>].

<перехід> ::= goto <мітка>.

<цикл> ::= for <присвоєння> to <константа> do <тіло\_програми> |

for <присвоєння> downto <константа> do <тіло\_програми> |

while <логічний\_вираз> do <тіло\_програми> |

repeat <тіло\_програми> until <логічний\_вираз>.

<ввід> ::= get(<ідентифікатор>);.

<вивід> ::= put(<арифметичний\_вираз> | <рядок>);.

<рядок> ::= "{<літера\_або\_цифра>}".

РБНФ Кока — Янгера — Касами

program = program\_part1, program\_part2 ;

program\_part1 = "program", program\_name, ";", "start", ( "var", declaration | "var" ) ;

program\_part2 = ";", ( statement\_iteration, "finish" | statement, "finish" | "finish" ) ;

program\_name = ident\_terminal ;

declaration = value\_type, ident, other\_declaration\_ident\_iteration ;

other\_declaration\_ident\_iteration = { ",", ident } ;

value\_type = "int\_16t" ;

lr\_expression = expression, "==>" ;

bind\_left\_to\_right = lr\_expression, ident ;

expression = left\_expression, [ binary\_action\_iteration ] ;

left\_expression = ( "(", expression, ")" )

| unary\_operator, expression

| ident\_terminal

| value\_terminal ;

binary\_action\_iteration = binary\_action, { binary\_action } ;

binary\_action = binary\_operator, expression ;

unary\_operator = "!" | "-" | "+" ;

binary\_operator = "and" | "or" | "==" | "!=" | "<=" | ">=" | "+" | "-" | "\*" | "/" | "%" ;

cond\_block = "if", "(", expression, ")", body\_for\_true, [ body\_for\_false ] ;

body\_for\_true = ( statement\_iteration, ";" | statement, ";" | ";" ) ;

body\_for\_false = ( "ELSE", statement\_iteration, ";"

| "ELSE", statement, ";"

| "ELSE", ";" ) ;

forto\_cycle = "for", cycle\_counter\_init, "to", cycle\_counter\_last\_value, cycle\_body, ";" ;

cycle\_counter\_init = (cycle\_counter, rl\_expression | lr\_expression, cycle\_counter) ;

cycle\_counter = ident\_terminal ;

cycle\_counter\_last\_value = value\_terminal ;

cycle\_body = "do", (statement\_in\_while\_body\_iteration | statement) ;

while\_cycle = "while", expression,

repeat\_until\_cycle = "repeat",

(statement\_iteration, "until", group\_expression |

statement, "until", group\_expression |

"until", group\_expression) ;

group\_expression = "(", expression, ")" ;

statement\_in\_while\_body\_iteration = statement\_in\_while\_body, statement\_in\_while\_body\_iteration | statement\_in\_while\_body ;

statement\_iteration = statement, statement\_iteration | statement ;

input = "get", "(", ident, ")" ;

output = "put", "(", expression, ")" ;

statement = ( ident, rl\_expression

| lr\_expression, ident

| cond\_block

| forto\_cycle

| while\_cycle

| repeat\_until\_cycle

| label, ":"

| "goto", label

| input

| output ) ;

statement\_iteration = statement, { statement } ;

statement\_in\_while\_body = statement ;

statement\_in\_while\_body\_iteration = statement\_in\_while\_body, { statement\_in\_while\_body } ;

2.2 **Опис термінальних символів та ключових слів**

- початок програми program<name>

- початок блоку даних var

- типи даних: int16\_t;

- оператор вводу: get;

- оператор виводу: put;

- блок тіла програми: start - finish

- 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: Up8;

- операції арифметичні: +, -, \*, /, %;

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

- операції логічні: !, and, or;

- коментар: ??...;

1. Розробка транслятора з вхідної мови програмування
   1. Вибір технології програмування.
   2. Проектування таблиць транслятора та вибір структур даних.
   3. Розробка лексичного аналізатора.
      1. Розробка алгоритму роботи лексичного аналізатора.
      2. Опис програми реалізації лексичного аналізатора.
   4. Розробка синтаксичного та семантичного аналізатора.
      1. Розробка дерева граматичного розбору.
      2. Розробка алгоритму роботи синтаксичного і семантичного аналізатора.
      3. Опис програми реалізації синтаксичного та семантичного аналізатора.
   5. Розробка генератора коду.
      1. Розробка алгоритму роботи генератора коду.
      2. Опис програми реалізації генератора коду.
2. Налагодження та тестування розробленого транслятора
   1. Опис інтерфейсу та інструкції користувачу.
   2. Виявлення лексичних і синтаксичних помилок.
   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.

program MPROGRAM

var

?? argumentValueA ??

int16\_t VERSIONA ?? int A; ??

?? argumentValueB ??

int16\_t VERSIONB ?? int B; ??

?? resultValues ??

int16\_t SUMSUMAB, DDIIFFAB, PRODDDAB, MODMODAB ?? int sum, diff, product, mod; ??

int16\_t DIVISIAB ?? float division; ??

int16\_t VALUEXAB, VALUEYAB ?? int X, Y; ??

;

start

?? input ??

get VERSIONA; ?? scanf("%d", &A); ??

get VERSIONB; ?? scanf("%d", &B); ??

?? compute ??

?? Perform addition ??

SUMSUMAB ==> VERSIONA + VERSIONB; ?? sum = A + B; ??

?? Perform subtraction ??

DDIIFFAB ==> VERSIONA - VERSIONB; ?? diff = A - B; ??

?? Perform multiplication ??

PRODDDAB ==> VERSIONA \* VERSIONB; ?? product = A \* B; ??

?? Perform division with float result ??

DIVISIAB ==> VERSIONA / VERSIONB; ?? division = (float)A / B; ??

?? Calculate remainder ??

MODMODAB ==> VERSIONA % VERSIONB; ?? mod = A % B; ??

?? Calculate expressions for X and Y ??

VALUEXAB ==> VERSIONA - VERSIONB \* 10 + VERSIONA + VERSIONB / 10; ?? X = (A - B) \* 10 + (A + B) / 10; ??

VALUEYAB ==> VALUEXAB + VALUEXAB % 10; ?? Y = X + X % 10; ??

?? output ??

put SUMSUMAB; ?? printf("%d", sum); ??

put DDIIFFAB; ?? printf("%d", diff); ??

put PRODDDAB; ?? printf("%d", product); ??

put DIVISIAB; ?? printf("%.2f", division); ??

put MODMODAB; ?? printf("%d", mod); ??

put VALUEXAB; ?? printf("%d", X); ??

put VALUEYAB; ?? printf("%d", Y); ??

finish

**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 (A=C) i (B=C)»), 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, якіщо число А більше за суму чисел В і С (логічний вираз в умовному операторі має виглядати так: «! (А<(В+С))»), інакше вивести 0.

program MPROGRAM

var

?? argumentValues ??

int16\_t FIRSTNAB

int16\_t SECONDAB

int16\_t THIRDNAB

?? resultValues ??

int16\_t MAXMAXAB, EQUALFAB, NEGATIAB, SUMFLAAB

start

?? input ??

get FIRSTNAB;

get SECONDAB;

get THIRDNAB;

?? compute ??

?? Find the maximum number using nested conditional statements ??

if FIRSTNAB >= SECONDAB

if FIRSTNAB >= THIRDNAB

MAXMAXAB ==> FIRSTNAB;

else

MAXMAXAB ==> THIRDNAB;

??endif??

else

if SECONDAB >= THIRDNAB

MAXMAXAB ==> SECONDAB;

else

MAXMAXAB ==> THIRDNAB;

??endif??

??endif??

?? Check if all numbers are equal ??

if FIRSTNAB eg SECONDAB and FIRSTNAB eg THIRDNAB and SECONDAB eg THIRDNAB

EQUALFAB ==> 1;

else

EQUALFAB ==> 0;

??endif??

?? Check if any number is negative ??

if FIRSTNAB <= 0 or SECONDAB <= 0 or THIRDNAB <= 0

NEGATIAB ==> -1;

else

NEGATIAB ==> 0;

??endif??

?? Check if FIRSTNAB is greater than the sum of SECONDAB and THIRDNAB ??

if ne |FIRSTNAB <= |SECONDAB + THIRDNAB||

SUMFLAAB ==> 10;

else

SUMFLAAB ==> 0;

??endif??

?? output ??

put MAXMAXAB;

put EQUALFAB;

put NEGATIAB;

put SUMFLAAB;

finish

**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:

X = 0

Цикл від 1 до А з кроком 1 Цикл від 1 до В з кроком 1 X = X + 1

1. Вивести значення Х на екран.

program SQUARESQ

var

??argumentValues??

int16\_t STARTNUM, ENDNUMBR ??int a, b;??

??resultValues??

int16\_t XXXXXXXX, SQUARNUM ??int x, square;??

;

start

??input??

get STARTNUM; ??scanf("%d", &a);??

get ENDNUMBR; ??scanf("%d", &b);??

??compute??

?? Output squares of numbers from STARTNUM to ENDNUMBR ??

for STARTNUM to ENDNUMBR ??for (int i = a; i <= b; i++)??

SQUARNUM ==> STARTNUM \* STARTNUM; ??square = i \* i;??

put SQUARNUM; ??printf("%d", square);??

STARTNUM ==> STARTNUM + 1; ??i++;??

finish

?? Initialize X to 0 ??

XXXXXXXX ==> 0; ??x = 0;??

?? Nested loop to calculate X = A \* B ??

for 1 to STARTNUM ??for (int i = 1; i <= a; i++)??

for 1 to ENDNUMBR ??for (int j = 1; j <= b; j++)??

XXXXXXXX ==> XXXXXXXX + 1; ??x = x + 1;??

finish

finish

??output??

put XXXXXXXX; ??printf("%d", x);??

finish

**Висновки**

Список літературних джерел

Додатки

А. Таблиці лексем для тестових прикладів

Б. С код (або код на асемблері), отриманий на виході транслятора для тестових прикладів;

В. Абстрактне синтаксичне дерево для тестових прикладів (за наявності).

Г. Документований текст програмних модулів (лістинги)