УДК 004.42

М.А. Бондаренко, В.А. Жилін, Д.П. Панасенко

*Українська інженерно-педагогічна академія, Харків*

# РОЗРОБКА КЛАСІВ ДЛЯ ПІДСИСТЕМ СИНТАКСИЧНОГО АНАЛІЗУ ТРАНСЛЯТОРІВ З АЛГОРИТМІЧНИХ МОВ ВИСОКОГО РІВНЯ

*В роботі розглядається розробка певних інструментальних засобів, які можуть бути використані при розробці підсистеми синтаксичного аналізу арифметичних виразів. Такими інструментальними засобами є нові класи в системі об'єктно-орієнтованого програмування. Розробку класів здійснено засобами системи ООП Delphi 6.*

***Ключові слова:*** *синтаксичний аналіз, об'єктно-орієнтоване програмування, класи даних.*

## Вступ

На підставі аналізу вимог до підсистеми синтаксичного аналізу в компіляторах мов програмування високого рівня розроблено її об'єктну модель. Це дало можливість розробити два нових класи, за допомогою яких побудовано модель синтаксичного аналізу арифметичних виразів. Розробка здійснена у середовищі об'єктно-орієнтованого програмування Delphi 6. Наведено результати розробки та тестування властивостей і методів нових класів, а також приклад їх використання в прикладному додатку.

Як відомо, розвиток комп'ютерної техніки привів до необхідності створення нового прикладного програмного забезпечення. Програмувати в машинних кодах (або на асемблері) стало істотно складніше — пропорційно зросли обсяги кодів програм, на написання яких були потрібні значні тимчасові витрати, а також ґрунтовні знання операційної системи.

Необхідність появи спеціалізованих середовищ поставили собі за мету розроблювачі систем програмування. Задача зводилася до полегшення операцій зі взаємодії з операційною системою за допомогою внутрішньої реалізації практично всіх детальних функцій, а також істотному спрощенню за рахунок часткової автоматизації самого процесу, що дозволяє різко скоротити терміни розробки програм і підвищити їх якість.

Так з'явилися мови програмування високого рівня, що є системами швидкої розробки додатків. Сучасні могутні мови програмування дозволяють створювати програмні продукти практично для будь-якої області сучасних комп'ютерних технологій: бізнес-додатки, мультимедіа, ігри, бази даних. При цьому додатки можуть бути як простими, так і складними, в залежності від поставленої задачі.

Але сама по собі мова програмування мало що дає розробнику додатків, якщо програму цією мовою неможливо виконати на комп'ютері. А для цього вона, врешті решт, має бути переведена в машинні коди. Цю задачу вирішує спеціальна програма – транслятор (або компілятор) з цієї мови в машинні коди.

Розробка компілятору з мови програмування високого рівня — дуже складна задача, яка вирішується чималими колективами програмістів за тривалий час. Це, насамперед, пов'язано зі складністю цієї задачі.

Так, наприклад, у 1996 році фірма Borland, яка відома своїми розробками в області реалізації мов програмування, випустила компілятор нового покоління Delphi. Насамперед, це могутній компілятор мови Pascal, яка доповнена новими істотними можливостями для створення додатків у середовищі Windows. Якісно новий тип інтерфейсу середовища (Visual) дозволяє при складанні програми бачити ті графічні об'єкти, для яких вона пишеться. Delphi – це система програмування високого рівня. Вона бере на себе значну частину роботи з керування комп'ютером, що уможливлює в простих випадках обходитися без особливих знань про деталі її роботи.

Зазвичай компілятор складається з декількох підсистем, зокрема, підсистеми лексичного аналізу і підсистеми синтаксичного аналізу. Задача цих підсистем — аналіз тексту програми, пошук помилок на етапі компіляції, підготовка до генерації машинного коду.

Отже, в даній роботі розглядається розробка нових класів у системі об'єктно-орієнтованого програмування, які можуть бути використані при створенні підсистеми синтаксичного аналізу арифметичних виразів. Розробку класів здійснено засобами системи ООП Delphi 6.

## Аналіз вимог до підсистеми синтаксичного аналізу

Розробка повної підсистеми синтаксичного аналізу для мови програмування високого рівня є складною задачею. Тому в даній роботі ми обмежимося розглядом синтаксичного аналізу арифметичних виразів.

 

У загальному вигляді структуру компілятора можна представити в такому вигляді (рис. 1).

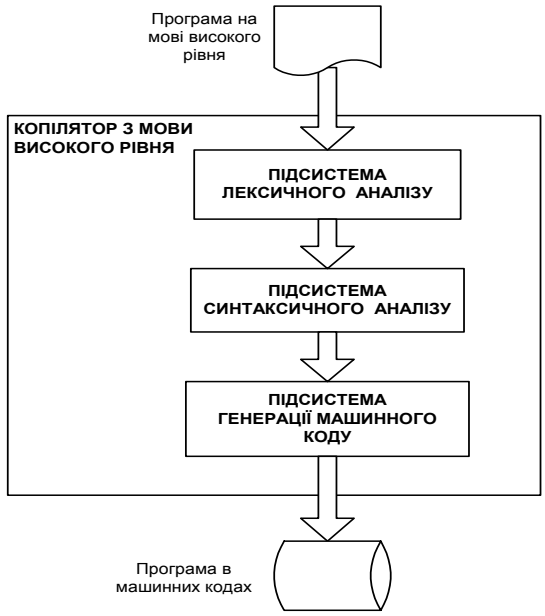


Рис. 1. Загальна структура компілятору

Головна задача лексичного аналізу — перевірка тексту програми на лексичні помилки і його переводу у послідовність лексем — спеціальних одиниць одного розміру, які містять інформацію про ідентифікатори, зарезервовані слова, знаки операцій тощо.

Саме ця послідовність лексем поступає на вхід підсистеми синтаксичного аналізу. Будемо моделювати лексеми латинськими буквами.

Як вже зазначалося, обмежимося розглядом синтаксичного аналізу арифметичних виразів. Головна задача цього аналізу — перевірка виразу на помилки і розбиття складного арифметичного виразу на послідовність відповідних простих арифметичних операцій: складання, віднімання, множення та ділення.

Будемо здійснювати перевірку двох типів помилок:

1. Баланс дужок. Помилка має виникати тоді, коли кількість відкриваючих дужок не дорівнює кількості закриваючих.

2. Пара операцій. У арифметичному виразі не допускається сполучення двох знаків операцій поряд (++, +–, *+ і т. п.).

Розбиття складного арифметичного виразу на послідовність відповідних простих арифметичних операцій має проводитися з урахуванням трьох чинників:

1) у відповідності до дужок, у тому числі вкладених;

2) у відповідності до пріоритетів операцій (спочатку – множення та ділення, а потім – складання та віднімання);

3) послідовність елементарних арифметичні операції, на які розбивається вираз, мають містити так звані тимчасові змінні, які беруть участь при обчисленні арифметичного виразу. Зазвичай при генерації машинного коду ці змінні є швидкими регістрами процесору, а елементарні операції – відповідними машинними командами.

Наприклад, вираз A=B+C*D має бути переведено в таку послідовність елементарних операцій:

R0=C*D R1=B+R0 A=R1

Тут R0, R1 – тимчасові змінні (регістри), які використовуються для при обчисленні складного арифметичного виразу.

Ще один приклад.

Вираз A=(B+C)*(D–E) має бути переведено в таку послідовність елементарних операцій:

R0=B+C R1=D–E R2=R1*R2 A=R2

Алгоритм такого розбиття арифметичного виразу складається з двох незалежних процедур. При цьому будемо виходити з припущення, що вираз не містить помилок.

Перша процедура – розбиття арифметичного виразу, який не містить дужок. Для цього у відповідності до пріоритетів операцій виконується пошук спочатку знаків множення та ділення і генерація елементарних операцій. Потім виконується пошук знаків складання та віднімання. Тимчасові змінні, які при цьому з’являються, заміщають у виразі ці операцію.

Наприклад, розглянемо вираз

A=B*C+D*E.

Спочатку знаходимо перший знак множення (*) і генеруємо першу елементарну операцію – R0=B*C. Ця тимчасова змінна підставляється у вираз замість цієї елементарної операції: A=R0+D*E. Таку заміну елементарного виразу тимчасовою змінною будемо називати спрощенням виразу.

Далі ця процедура повторюється і ми отримуємо елементарну операцію R1=D*E і модифікацію початкового виразу: A=R0+R1.

Повторюємо процедуру. Отримуємо елементарну операцію R2=R0+R1 і остаточну модифікацію виразу: A=R2.

Друга процедура – розбиття арифметичного виразу, який містить дужки. Цей алгоритм такий. Спочатку виконується пошук закриваючої дужки (справа наліво). Потім з цієї позиції виконується пошук першої відкриваючої дужки (зліва направо). Арифметичний вираз (не обов’язково елементарний) між цими дужками не містить дужок і до нього може бути застосовано перша процедура. Отримана в результаті роботи першої процедури тимчасова змінна підставляється у початковий вираз замість дужок і виразу в них.

Наприклад, розглянемо вираз

A=B*(C+D*(K+L/M)).

Пошук першої закриваючої дужки і відповідної відкриваючої дає вираз K+L/M, який не містить дужок. Після застосування першої процедури отримуємо послідовність елементарних операцій:

R0=L/M R1=K+R0.

Модифікація початкового виразу:

A=B*(C+D*R1).

Повторюємо пошук відповідних дужок і отримаємо:

R2=D*R1 R3=C+R2.

Модифікація початкового виразу:

A=B*R3.

Цей випадок вже було розглянуто в першій процедурі.

Такі основні алгоритми синтаксичного аналізу арифметичних виразів.

## Розробка об'єктної моделі підсистеми синтаксичного аналізу арифметичних виразів

У відповідності до розглянутих алгоритмів синтаксичного аналізу арифметичних виразів було запропоновано два класи, за допомогою яких зручно розробляти відповідну програму.

**І. Клас «Допоміжна таблиця».** Цей клас призначено для запису та зберігання елементарних операцій, на яке буде розбито арифметичний вираз. Кожний рядок цієї таблиці містить одну таку операцію.

Властивості класу: Count – кількість рядків у таблиці. Режим доступу – тільки читання.

Методи класу:

1. Add – додавання елементарної операції до таблиці.

2. Get(i:integer) – отримання інформації про і-ту елементарну операцію.

**ІІ. Клас «Арифметичний вираз»**.

Властивості класу: Content – рядок, який містить арифметичний вираз. Режим доступу – читання та запис.

Методи класу:

1. Analyze – перевірка арифметичного виразу на помилки. Це функція, яка повертає значення 0, якщо помилок немає та значення 1, якщо помилки є.

2. Simplify – спрощення арифметичного виразу. Параметром цієї процедури є тимчасова таблиця, в яку будуть записані відповідні елементарні операції.

## Програмування методів класу «Арифметичний вираз»

Виходячи з запропонованої об'єктної моделі, було розроблено такий програмний код для класу «Арифметичний вираз».

```
unit ClassAV;
interface
uses
    Windows, Messages, SysUtils, Variants,
Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, DateUtils,ClassTempTable;
{Визначення класу «Арифметичний вираз»}
  type
  TAV = class
  private
{Внутрішні змінні}
    sContent: string;    // Арифметичний вираз
{Процедури та функції доступу до властивостей}
    function GetContent: string;
    procedure SetContent(C: string);
    procedure NoBrackets(var V:string; var
T:TTempTable);
  public
{Властивості класу}
{Арифметичний вираз}
    property Content: string read GetContent write
SetContent;
{Методи класу}
{Аналіз арифметичного виразу}
    function Analyze:integer;
{Спрощення арифметичного виразу}
    procedure Simplify(T:TTempTable);
  end;
  implementation
{Реалізація властивостей}
{Читання арифметичного виразу}
  function TAV.GetContent: string;
  begin
    Result:= sContent;
  end;
{Запис арифметичного виразу}
  procedure TAV.SetContent(C: string);
  begin
    sContent:=C;
  end;
{Реалізація методів}
{Аналіз арифметичного виразу}
  function TAV.Analyze:integer;
  var
    i:integer;
    no:integer;  // Кількість дужок (
    nz:integer;  // Кількість дужок )
    po:integer;  // Позиція дужки (
    s:string;
  begin
    // Перевірка балансу дужок
    no:=0;
    nz:=0;
    for i:=1 to Length(sContent) do
    begin
      case sContent[i] of
      '(':
        begin
          no:=no+1;
          po:=i;
        end;
      ')':
```

```
            begin
              nz:=nz+1;
              if nz>no then
              begin
                Result:=-1;
                exit; end; end; end; end;
      if no<>nz then
      begin
        Result:=-1;
        exit;
      end
      else Result:=0;
      if no+nz<>0 then exit;
      // Перевірка пар лексем
      for i:=1 to Length(sContent)-1 do
      begin
        case sContent[i] of
        '+','-','*','/':
        begin
          case sContent[i+1] of
          '+','-','*','/':
          begin
            Result:=-1;
            exit; end; end; end; end; end;
      // Дужки відсутні. Перевірка операцій
      if Length(sContent)<=5 then Result:=0 else
Result:=1;
    end;
{Спрощення арифметичного виразу}
    procedure TAV.Simplify(T:TTempTable);
    var
      i,k:integer;
      no:integer;  // Кількість дужок (
      nz:integer;  // Кількість дужок )
      po:integer;  // Позиція дужки (
      s:string;
    begin
      // Зміна виразу у дужках
      no:=0;
      nz:=0;
      repeat
      po:=0;
      // Пошук дужок
      for i:=1 to Length(sContent) do
      begin
        case sContent[i] of
        '(':  po:=i;
        ')':
          begin
            s:=copy(sContent,po+1,i-po-1); // Вибір
виразу в дужках
            NoBrackets(s,T);
            k:=ord('A')+T.Count;  // Формування
допоміжної змінної
            s:=chr(k)+'='+s;      // Формування
операції
            T.Add(s);             // Додавання
операції до таблиці
            Delete(sContent,po,i-po+1); // Вилу-
чення виразу та дужок
            Insert(chr(k),sContent,po); // Заміна на
допоміжню змінну
            break; end; end; end;
      until po=0;
      NoBrackets (sContent,T);
    end;
    procedure TAV.NoBrackets(var V:string; var
T:TTempTable);
     // Дужки відсутні. Подальше спрощення вира-
зу
    var
      i,k:integer;
      po:integer;  // Позиція операції
      s:string;
    begin
      // Обробка множення та ділення
      repeat
      for i:=1 to Length(V) do
      begin
        po:=0;
        case V[i] of
        '*','/':
          begin
            po:=po+1;
            s:=copy(V,i-1,3); k:=ord('A')+T.Count;
            s:=chr(k)+'='+s; T.Add(s);
            Delete(V,i-1,3); Insert(chr(k),V,i-1);
            break; end; end; end;
      until po=0;
      // Обробка додавання та віднімання
      for i:=1 to Length(V) do
      begin
        case V[i] of
        '+','-':
          begin
            s:=copy(V,i-1,3); k:=ord('A')+T.Count;
            s:=chr(k)+'='+s; T.Add(s);
            Delete(V,i-1,3); Insert(chr(k),V,i-1);
            break; end; end; end; end.
```

## Програмування методів класу «Допоміжна таблиця»

Виходячи із запропонованої об’єктної моделі, було розроблено такий програмний код для класу «Допоміжна таблиця».

```
    unit ClassTempTable;
    interface
    uses
     Windows, Messages, SysUtils, Variants,
Classes, Graphics, Controls, Forms,
      Dialogs, StdCtrls, DateUtils;
     {Визначення класу «Допоміжня таблиця»}
    type
    TTempTable = class
    private
     {Внутрішні змінні}
```

```
        aT: array [1..100] of string;    // Для
зберігання допоміжньої таблиці
        iCount: integer;                 // Для зберігання
кількості рядків в таблиці
       {Процедури та функції доступу до власти-
востей}
        function GetCount: integer;  // Тільки читання
      public
       {Властивості класу}
        {Кількість рядків в таблиці}
        property Count: integer read GetCount;  //
Тільки читання
        {Методи класу}
        {Додавання до таблиці}
         procedure Add (s:string);
        {Длступ до рядків таблиці}
         function Get(i:integer):string;
      end;
      implementation
        {Реалізація властивостей}
        {Підрахування кількості рядків в таблиці}
      function TTempTable.GetCount: integer;
      begin
         Result:= iCount;
      end;
        {Реалізація методів}
         {Додавання до таблиці}
      procedure TTempTable.Add (s:string);
      begin
          iCount:=iCount+1;
          aT[iCount]:=s;
      end;
        {Доступ до рядків таблиці}
      function TTempTable.Get(i:integer):string;
      begin
          if (i<=0) or (i>iCount) then
          begin
              ShowMessage('ПОМИЛКА ДОСТУПУ ДО
ТАБЛИЦІ');
              result:='';
              exit; end;
          result:=aT[i];
      end; end.
```

## Основні результати роботи:

1. На підставі аналізу вимог до функціонування підсистеми синтаксичного аналізу арифметичних виразів побудовано об'єктну модель цієї підсистеми. Визначено два класи — «Допоміжна таблиця» і «Арифметичний вираз», за допомогою яких можна здійснити програмну реалізацію підсистеми синтаксичного аналізу.

2. Здійснена програмна реалізація класів TAV («Арифметичний вираз») і TTempTable («Допоміжна таблиця»).

Нові класи можуть бути корисним для розробників компіляторів з мов програмування високого рівня.

## Список літератури

*1. Бобровский С. Delphi 5. Учебный курс / С. Бобровский. – СПб.: Питер, 2007. – 640 с.*

*2. Культин Н. Программирование в Delphi 5 / Н. Культин. – СПб.: Питер, 2000. – 464 с.*

*3. Фаронов В.В. Delphi 5: Учебный курс / В.В. Фаронов. – М.: Нолидж, 2000. – 608 с.*

*4. Бондаренко М.А. Програмування у середовищі Delphi / М.А. Бондаренко. – Х.: ФОП Лібуркіна Л.М., 2007. – 600 с.*

*Надійшла до редколегії 27.04.2010*

**Рецензент:** д-р техн. наук, проф. В.А. Краснобаєв, Харківський національний технічний університет сільського господарства ім. П. Василенка, Харків.

### РАЗРАБОТКА КЛАССОВ ДЛЯ ПОДСИСТЕМ СИНТАКСИЧЕСКОГО АНАЛИЗА ТРАНСЛЯТОРОВ С АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ

Н.А. Бондаренко, В.А. Жилин, Д.П. Панасенко

*В работе рассматривается разработка определенных инструментальных средств, которые могут быть использованные при разработке подсистемы синтаксического анализа арифметических выражений. Такими инструментальными средствами являются новые классы в системе объектно-ориентированного программирования. Разработка классов осуществлено средствами системы ООП Delphi6.*

***Ключевые слова:*** *синтаксический анализ, объектно-ориентированное программирование, классы данных.*

### DEVELOPMENT OF CLASSES FOR THE SUBSYSTEMS OF SYNTACTIC ANALYSIS OF TRANSLATORS FROM ALGORITHMIC LANGUAGES HIGH LEVEL

N.A. Bondarenko, V.A. Zhilin, D.P. Panasenko

*Development of certain tools which can be the subsystems of syntactic analysis of arithmetic expressions used for development is examined. Such tools are new classes in the system of the object-oriented programming. Development of classes it is carried out facilities of the system of OOP Delphi6.*

***Keywords:*** *syntactic analysis, object-oriented programming, classes of information.*