УДК 004.43

# МЕТОДЫ РАСШИРЕНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ (Часть 1)

***В. Д. Михеева*[1],**
*старший инженер по программному обеспечению*
*Российское отделение компании «Интел»*

*Приводится обзор методов расширения современных языков программирования, определенных автором и использованных для построения классификации расширений по способам интеграции и исполнения кода расширений. Рассматривается применение расширений в реальных системах программирования с примерами исходного кода на расширенных языках, анализируются достоинства и недостатки каждого из обсуждаемых методов. Рассмотрены наиболее простые и часто использующиеся методы интеграции кода расширений в основной код, не требующие изменений в базовой системе программирования.*

***Ключевые слова*** *— предметно-ориентированный язык программирования, расширение языка программирования, инструментальные средства программирования.*

## Введение

В современных проектах разработки программного обеспечения довольно часто возникает потребность в создании специализированного инструмента программирования для решения задач, связанных с определенной предметной областью [1]. Современные языки программирования общего назначения (например, C, C++, Object Pascal, Java) предоставляют разработчику широкий спектр возможностей, однако их применение в определенной предметной области может оказаться неудобным и неэффективным. То есть универсального языка программирования, одинаково результативного для применения в любой области, не существует [1–4]. В этом случае наиболее действенным является применение специально предназначенных для узкого класса задач предметно-ориентированных языков [2]. Однако разработка специализированного языка программирования является трудоемким и затратным процессом, не всегда возможным в условиях отдельно взятого проекта. Кроме того, зачастую, наряду со специализированными возможностями, от языка программирования одновременно требуется широкий спектр функциональных возможностей, присущих языкам общего назначения [3]. Но не нужно торопиться изобретать велосипед — достаточно его усовершенствовать. В результате многолетней истории создания языков программирования появились развитые языки общего назначения, обладающие многими достоинствами, предоставляющие всю необходимую базовую функциональность и имеющие богатую инструментальную поддержку. В силу этих причин вместо разработки совершенно нового языка программирования во многих случаях лучшим решением является применение различных способов совмещения кода на основном языке программирования общего назначения с кодом специализированных расширений, другими словами, — реализация расширения уже существующего языка программирования. Данный обзор сделан с целью сформировать представление о многообразии специализированных средств в современных языках и системах программирования, а также сориентировать разработчика ПО в выборе подходящих методов при необходимости реализовать собственные расширения языков программирования. В общей сложности автором определено восемь методов расширений языков программирования: четыре метода интеграции кода расширений в языки программирования и четыре метода исполнения кода расширений. Кроме этого, в последнем разделе приводится пример предметно-ориентированного расширения традиционного языка программирования средствами решения задач с данными в таб-

[1] Научный руководитель — кандидат физ.-мат. наук, заведующий лабораторией астрономического программирования Института прикладной астрономии РАН *Ф. А. Новиков*.

личной форме [5], с обоснованием выбора подходящих в этом случае методов расширения базового языка (БЯ). Данное расширение реализовано автором [6] на базе средств программирования системы ЭРА (эфемеридные расчеты в астрономии) [7] и применяется для решения задач эфемеридной астрономии.

## Классификация расширений по методу их интеграции в базовый язык

По мнению автора, в общем случае, согласно имеющейся практике, можно выделить несколько основных способов совмещения (интеграции) разных языковых конструкций в одной программе и классифицировать в соответствии с ними расширения современных языков программирования. Предлагается также использовать термин *метод расширения* (более конкретно, *метод интеграции расширения* или *метод исполнения расширения*) применительно к каждому из таких способов. Классификация расширений по методу их интеграции в БЯ представляет собой список следующих категорий (здесь они приведены в порядке усиления выразительности языковых средств, использующихся для описания специализированных расширений).

1. Расширенные возможности реализуются отдельно, обычно на другом языке, и доступны в программе на БЯ через интерфейс API.

2. Расширения БЯ реализованы на том же языке и доступны через определенное подмножество БЯ.

3. Конструкции другого языка вносятся в текст программы на БЯ «как есть», но в виде строкового значения, передающегося в качестве параметра в интерфейсе API для его интерпретации.

4. Базовый язык расширен новыми языковыми конструкциями.

Заметим, что в реальных системах программирования для расширения языковых возможностей редко применяется какой-то один метод. Чаще всего используется комбинация двух и более методов.

## Метод 1. Расширения доступны через API

Это простейший (прямолинейный) способ введения расширений в БЯ в том смысле, что расширение выразительных возможностей БЯ не происходит (но появляются новые функциональные возможности) — расширенные возможности вводятся в текст программы обычными вызовами функций, подпрограмм, манипуляцией с объектами, модулями, т. е. посредством интерфейса API, выраженного в конструкциях БЯ. Реализация же самих расширенных возможностей выполняется отдельно от основной программы, чаще всего на другом языке программирования, и предоставляется в виде готового исполняемого кода, как правило, без исходного текста — в виде объектных модулей библиотек или DLL (а в некоторых случаях встраивается в систему исполнения — run-time system). В этом случае код реализации расширений связывается вместе с объектным кодом основной программы статически, на этапе сборки приложения компоновщиком (linker), а в случае реализации расширений в виде DLL — динамически, путем вызовов функций из DLL во время исполнения программы средствами операционной системы. Это достаточно распространенный способ введения расширенных возможностей в язык программирования общего назначения, и мы убедимся в этом, рассмотрев несколько примеров, приведенных в данном разделе. В качестве недостатков данного метода следует отметить ограниченность выразительных средств расширений, представленных лишь внешними переменными и функциями, а также накладные расходы на исполнение внешних функций и передачу параметров, порой существенные.

***Пример: стандартные библиотеки C++.***

Реализация стандартных библиотек C++ [8] выполнена по описанному выше принципу: функциональность библиотек реализована разработчиком компилятора C++ (например, компанией Microsoft) специально для целевой платформы и поставляется отдельным компонентом в виде исполняемого кода для компоновки с текстом пользовательской программы, использующей его через интерфейс API. Чтобы применить функциональность этих библиотек, пользователь в своей программе должен вставить ссылку на специальные модули, содержащие описание интерфейса API. Тогда необходимые модули подключатся к исходному коду пользовательской программы компилятором и компоновщиком. Модули с описанием интерфейса API могут быть доступны в виде исходного кода на том же БЯ программирования (в данном случае, на C++) или опять же лишь в бинарном виде. Реализация функциональности стандартных библиотек C++ может быть выполнена частично на C++, частично на ассемблере (т. е. не на C++) или в виде бинарного кода. Реализация стандартных библиотек C++ зависит от целевой платформы и у каждого разработчика компилятора – своя.

***Пример: язык VBA и библиотека DAO.***

Рассмотрим другой пример. В Microsoft Access есть возможность программирования на языке Visual Basic for Applications (VBA) для автоматизации манипуляций с объектами баз данных (БД). В среде программирования на VBA есть

условия для подключения *объектных модулей библиотек* с реализацией разных дополнительных возможностей, расширяющих функциональность программ на VBA. Например, можно подключить входящую в стандартную поставку Microsoft Access библиотеку Объектов доступа к данным (Data Access Objects — DAO), предоставляющую *объектно-ориентированный интерфейс* API для типичных операций с таблицами и запросами в реляционной БД. В примере 1 [9, гл. 16, программа 16.5] показано использование интерфейса библиотеки DAO в программе на VBA для изменения структуры таблицы в реализации приложения «Игра в доминирование». Строки пронумерованы для удобства рассмотрения программного кода.

*Пример 1. Программа на VBA с использованием библиотеки DAO.*

**Программа 16.5. Изменение структуры таблицы с помощью интерфейса DAO**

```
 1 Dim db As Database
 2 Dim fieldSize As Long, i As Long
 3 Dim strSQL As String
 4 Dim fld As Field
 5 ' Узнаем линейный размер игрового поля, выраженный в клетках
 6 fieldSize = CLng(get_parameter("РазмерПоля"))
 7 ' Открываем базу данных, в которой хранится нужная таблица
 8 Set db = OpenDatabase(CurrentProject.Path & DominationGame.mdb")
 9 ' Удалить старые записи в таблице «ПолеИгрок»
10 StrSQL = "DELETE * FROM ПолеИгрок;"
11 db.Execute strSQL
12 ' Удалить старые поля в таблице «ПолеИгрок»
13 For i = 1 To db.TableDefs("ПолеИгрок").Fields.Count
14  db.TableDefs("ПолеИгрок").Fields.Delete get_column_name(i)
15 Next i
16 ' Создать новые поля в таблице «ПолеИгрок»
17 For i = 1 To fieldSize
18  Set fld = db.TableDefs("ПолеИгрок").CreateField( _
19                get_column_name(i), dbText, 20)
20  db.TableDefs("ПолеИгрок").Fields.Append fld
21 Next i
22 ' Прочие действия ...
23 db.Close
```

В этом примере проиллюстрировано удаление и добавление полей в таблицу "ПолеИгрок" с использованием интерфейса DAO. В книге [9, гл. 16] подробно описано, какие элементы интерфейса DAO (объекты и методы) использовались в коде удаления полей (строки 13–15) и в коде создания новых полей (строки 17–21): «Удаление поля производится с помощью метода Delete, в качестве параметра которого указывается имя удаляемого поля. Добавление нового поля производится следующим образом. Создается новый объект Field, обладающий необходимыми характеристиками — заданным именем поля, типом и размером данных. После этого с помощью метода Append созданный объект Field добавляется в семейство Fields объекта TableDef, содержащее все поля таблицы "ПолеИгрок"».

***Пример: язык общего назначения и доступ к БД.***

Подобным образом организована возможность работы с БД и в других средах программирования. Например, в Borland Delphi 6.0 для работы с БД в программах на Object Pascal можно использовать библиотеку BDE, а при программировании в Microsoft Visual Studio 2005 — библиотеку ADO.NET с объектно-ориентированным интерфейсом API. Для доступа к интерфейсу ADO.NET в программе на языке C#, например, необходимо подключить модуль System.Data, реализация которого поставляется в виде библиотеки динамической загрузки system.data.dll. В Microsoft Visual Studio 2005 такая же возможность поддерживается и для программ на языках C++, J#, Jscript, Visual Basic (VB). Рассмотрим пример программного кода на C# с использованием библиотеки ADO.NET [10].

*Пример 2. Программа на C# с использованием библиотеки ADO.NET.*

```
 1 using System;
 2 using System.Data;
 3 using System.Data.SqlClient;
 4
 5 namespace Microsoft.AdoNet.DataSetDemo
 6 {
 7   class NorthwindDataSet
 8   {
 9     ...
10     private static void ConnectToData(string connectionString)
11     {
12       // Create a SqlConnection to the Northwind database.
13       using (SqlConnection connection =
14           new SqlConnection(connectionString))
15       {
16         // Create a SqlDataAdapter for the Suppliers table.
17         SqlDataAdapter adapter = new SqlDataAdapter();
18
19         // A table mapping names the DataTable.
20         adapter.TableMappings.Add("Table", "Suppliers");
21
22         // Open the connection.
23         connection.Open();
24         Console.WriteLine("The SqlConnection is open.");
25
26         // Create a SqlCommand to retrieve Suppliers data.
27         SqlCommand command = new SqlCommand(
28           "SELECT SupplierID, CompanyName FROM dbo.Suppliers;",
29           connection);
30         command.CommandType = CommandType.Text;
31
32         // Set the SqlDataAdapter's SelectCommand.
33         adapter.SelectCommand = command;
34
35         // Fill the DataSet.
36         DataSet dataSet = new DataSet("Suppliers");
37         adapter.Fill(dataSet);
38
39         // Create a second Adapter and Command to get
40         // the Products table, a child table of Suppliers.
41         SqlDataAdapter productsAdapter = new SqlDataAdapter();
```

```
42      productsAdapter.TableMappings.Add("Table", "Products");
43
44      SqlCommand productsCommand = new SqlCommand(
45        "SELECT ProductID, SupplierID FROM dbo.Products;",
46        connection);
47      productsAdapter.SelectCommand = productsCommand;
48
49      // Fill the DataSet.
50      productsAdapter.Fill(dataSet);
51
52      // Close the connection.
53      connection.Close();
54
55      // Create a DataRelation to link the two tables
56      // based on the SupplierID.
57      DataColumn parentColumn =
58        dataSet.Tables["Suppliers"].Columns["SupplierID"];
59      DataColumn childColumn =
60        dataSet.Tables["Products"].Columns["SupplierID"];
61      DataRelation relation =
62        new System.Data.DataRelation("SuppliersProducts",
63          parentColumn, childColumn);
64      dataSet.Relations.Add(relation);
65      Console.WriteLine(
66        "The {0} DataRelation has been created.",
67        relation.RelationName);
68    }
69  }
70 }
71 }
```

В этом примере создается соединение с БД с помощью объекта типа SqlConnection (строки 13, 14), затем через специальный объект SqlDataAdapter устанавливается доступ к таблице «Поставщики» (Suppliers) и загружается набор записей в объект типа DataSet (строки 17–37). Далее аналогичным образом в этот же набор записей подгружается содержимое таблицы «Продукты» (Products) (строки 41–50). После этого с помощью метода Add коллекции ссылок Relations объекта типа DataSet создается ссылка на таблицу «Продукты» из таблицы «Поставщики» по ключевому полю «Код-Поставщика» (SupplierID) (строки 57–64). Из этого примера видно, что при всем удобстве объектно-ориентированного интерфейса к дополнительным возможностям управления БД из программы на языке общего назначения программирование таким способом достаточно утомительно. Оно трудоемко и требует написания большого количества строк кода только с подготовительными действиями (строки 13–53, 57–63), несмотря на то, что целью является выполнение над БД достаточно простого действия (строка 64). Просматривать, отлаживать и модифицировать такой длинный фрагмент кода неудобно.

***Пример: расширения, встроенные в систему исполнения.***

Выше мы рассмотрели случаи, когда расширения реализуются заранее и предоставляются пользователю вместе с инструментами программирования как отдельный компонент для встраивания в код пользовательской программы на этапе сборки или исполнения (в виде DLL). Но бывает, что подобная реализация расширений (выполненная отдельно и доступная через API) встроена в систему исполнения, например в интерпретатор приложений. Именно таким способом в системе ЭРА [7] выполнена интеграция специализированных предметно-ориентированных функций (действий и функций преобразования данных): их функциональность заранее реализована на языке Object Pascal и встроена в инструментальное средство *процессор языка СЛОН*, выполняющее трансляцию и интерпретацию программ на языке СЛОН (см. ч. 2 статьи).

***Пример: совмещение программ на разных языках через API.***

До сих пор мы говорили о случае, когда расширенные возможности предоставляются пользователю (программисту) в составе средств программирования. Однако у рассматриваемого метода совмещения реализаций программного кода на разных языках через интерфейс API есть еще один вариант использования: когда в качестве расширения пользовательской программы используется тоже пользовательский программный код (написанный этим же или другим программистом) на другом языке. Это легко представить в случае, когда один или несколько программистов разрабатывают части общего приложения на разных языках программирования и хотят соединить эти части вместе. В частности, распространенным примером является потребность совмещения программного кода, разработанного на разных языках, в целях миграции программного обеспечения, разработанного некоторое время назад, например на языке FORTRAN, Pascal или C. Такой код может понадобиться встроить, например, в программу на C++. Для этого в языке C++ предусмотрена специальная расширенная декларация внешней компоновки со спецификатором того языка, на котором реализована внешняя функциональность. В частности, на C++ декларации внешнего связывания в стиле C будут выглядеть следующим образом [11].

*Пример 3. Программа на C++ с внешним связыванием в стиле C.*

```
// Объявление функции printf с внешним связыванием в стиле C.
extern "C" int printf( const char *fmt, ... );
// Объявление глобальной переменной errno с внешним
 связыванием в стиле C.
extern "C" int errno;
```

Отметим, что при использовании внешнего связывания с функциями на другом языке программирования требуется соблюдать соответствующие соглашения по передаче параметров и включать необходимые опции сборки.

### Метод 2. Расширения в виде подмножества базового языка

Это достаточно простой технически способ внедрения расширений в БЯ общего назначения — реализовать специализированные возможности на том же БЯ и предоставить их пользователю (программисту) в виде подмножества БЯ, наделенного предметно-ориентированной семантикой и определенного набором правил (формальных требований), которым должен следовать пользователь для работы с данными специальными возможностями. Однако этот метод не всегда эффективен. В случаях, когда семантика расширений и семантика основного языка достаточно сильно или полностью несовместимы, выбор данного метода может повлечь введение таких ограничений на употребление выразительных средств БЯ, которые приведут к неудобству программирования, к увеличению объема ручного кодирования и, как правило, к неэффективности исполнения такого кода. А для исходного кода будут характерны плохая читабельность и трудность сопровождения.

***Пример: язык SystemC.***

Примером эффективного использования рассматриваемого метода введения расширений является язык SystemC.

Язык SystemC, созданный на основе языка C++ и специальных библиотек классов C/C++, предоставляет средство описания параллельных вычислений в целях построения моделей программно-аппаратных комплексов с различной степенью точности. Это средство широко используется в разработке систем на кристалле (System-on-Chip — SoC)[2]. В стандарте SystemC определен набор правил и ограничений, которым должен следовать разработчик программ на C++, чтобы получить правильный код на SystemC, а также разработчик реализации библиотеки SystemC. В библиотеку SystemC входит набор классов, функций и макросов, позволяющих оперировать в программе на C++ специализированными (предметно-ориентированными) понятиями, определяющими составные элементы (building blocks) моделируемой системы. В качестве примера рассмотрим фрагмент кода из документации по SystemC [14].

*Пример 4. Программа на C++ с использованием библиотеки SystemC.*

```
// Декларация модуля, описывающего устройство «Инкремент»
SC_MODULE(Increment) {
  sc_in<sc_int<16> > A;
  sc_out<sc_int<16> > X;
  const int m_delta; // инициализируется в конструкторе класса
  SC_HAS_PROCESS(Increment);

  Increment(sc_module_name& name_, int delta)
  : sc_module(name_), m_delta(delta)
  {
    SC_METHOD(proc);
    sensitive << A;
  }

  void proc() {
    X = A.read() + m_delta;
  }
};
```

В данном фрагменте кода описана архитектура и функциональность асинхронного устройства, выполняющего инкремент входного сигнала при каждом его изменении. Благодаря нотации SystemC в исходном коде наглядно представлены понятия из области описания устройств цифровой обработки сигналов. Предоставляемая OSCI реализация библиотек SystemC характеризуется эффективным исполнением программ на SystemC. Дополнительным плюсом данной реализации, поскольку пример 4 написан на синтезируемом подмножестве SystemC, является то, что рассматриваемый код пригоден для дальнейшей обработки инструментальными средствами синтеза — важной стадии дизайна аппаратного обеспечения.

И все-таки, хотя данный метод и приближает нас к желаемому выражению расширенных возможностей в терминах целевой задачи в рамках единой системы программирования, ему присущи общие для первых двух методов недостатки: избыточность программного кода, вытекающая из ограничений выразительных средств базовой системы программирования; отсутствие языкового контроля; слабая поддержка редактором исходного текста. Эти недостатки были отмечены еще в статье о парадигме LOP [2].

### Метод 3. Расширения как строковые параметры API

В этом случае программный код на специализированном языке записывается в виде строкового значения в тексте программы на БЯ программирования, и эта строка передается как параметр через интерфейс API доступа к библиотеке реализации расширений. Например, так мож-

[2] Создание и развитие SystemC — результат деятельности независимой некоммерческой ассоциации Open SystemC Initiative (OSCI) [12], в которую входят многие крупные компании индустрии электроники. Эта организация была основана в 1999 г. В декабре 2005-го SystemC был принят в качестве стандарта IEEE и опубликован в виде формального описания [13]. Реализация библиотеки SystemC может быть выполнена независимым разработчиком на основе описания в стандарте [13]. Ассоциацией OSCI предоставляется собственная реализация библиотеки SystemC, а также множество примеров, тестов, статей и других материалов.

но записать запрос на языке SQL и использовать его в качестве параметра объекта DAO в программе на VBA в Microsoft Access (см. пример 1, строки 10, 11) или аналогичным образом исполнить инструкцию на языке DDL, как показано в примере 5 [9, гл. 16, программа 16.4]. В строке 9 создана простая инструкция DDL удаления таблицы «ПолеИгрок», а в строке 12 она исполнена с помощью метода Execute объекта типа Database. Далее, в строках 16–20 сформирована более сложная инструкция DDL создания таблицы «ПолеИгрок» с новой структурой, включающая параметры, взятые из основной программы, — имена полей таблицы, полученные с помощью функции get_column_name(). Эта инструкция исполнена в строке 21.

*Пример 5. Программа на VBA с использованием строк языка DDL.*

**Программа 16.4. Удаление и создание таблицы с помощью инструкций DDL**

```
1  Dim db As Database
2  Dim fieldSize As Long, i As Long
3  Dim strDDL As String
4  ' Узнаем линейный размер игрового поля, выраженный в клетках
5  fieldSize = CLng(get_parameter("РазмерПоля"))
6  ' Открываем базу данных, в которой хранится нужная таблица
7  Set db = OpenDatabase(CurrentProject.Path &
"DominationGame.mdb")
8  ' Удалить старую таблицу «ПолеИгрок»
9  strDDL = «DROP TABLE ПолеИгрок;»
10 ' Если таблица не существует, удаление вызовет ошибку
11 On Error GoTo the_next_2
12 db.Execute strDDL
13 the_next_2:
14 On Error GoTo 0
15 ' Создать новую таблицу «ПолеИгрок»
16 strDDL = "CREATE TABLE ПолеИгрок ("
17 For i = 1 To fieldSize – 1
18  strDDL = strDDL & get_column_name(i) & " TEXT(20), "
19 Next i
20 strDDL = strDDL & get_column_name(fieldSize) & " TEXT(20) );"
21 db.Execute strDDL
22 ' Прочие действия
23 ...
24 db.Close
```

Коротко о языке DDL: «Для изменения схемы данных создан *язык определения данных* (DDL, Data-Definition Language). Инструкции на языке DDL позволяют выполнять действия по изменению схемы данных и структуры объектов данных, например:

- создавать и удалять таблицы;
- добавлять и удалять из таблиц поля и индексы;
- создавать и удалять связи между таблицами.

В интерфейсе DAO имеется набор специальных объектов, который является интерфейсом для доступа к средствам DDL. Таким образом, с помощью объектов DAO можно управлять структурой таблиц и схемой данных, не составляя самих инструкций на языке DDL. [...Другими словами] Аналогичные действия можно выполнить, пользуясь объектно-ориентированным интерфейсом DAO» [9, гл. 16].

Рассмотрим еще один пример использования строки запроса SQL в программе на языке общего назначения — на этот раз на языке C# [15].

*Пример 6. Программа на C# с использованием строки запроса SQL.*

```
1  SqlConnection c = new SqlConnection(...);
2  c.Open();
3  SqlCommand cmd = new SqlCommand(
4   @"SELECT c.Name, c.Phone
5     FROM Customers c
6     WHERE c.City = @p0");
7  cmd.Parameters.AddWithValue("@p0", "London");
8  DataReader dr = c.Execute(cmd);
9  while (dr.Read()) {
10    string name = dr.GetString(0);
11    string phone = dr.GetString(1);
12    DateTime date = dr.GetDateTime(2);
13 }
14 dr.Close();
```

В этом примере сначала формируется строка запроса SQL с параметром @p0 (строки 3–6), затем подставляется нужное значение параметра (строка 7), далее запрос исполняется (строка 8), и, наконец, полученный в результате набор данных подвергается обработке (строки 9–13).

Преимуществом рассматриваемого метода введения расширений является возможность записать в виде строки любую конструкцию расширения, в том числе даже не имеющую ничего общего с конструкциями БЯ, и при этом для поддержки работы таких расширений не нужно вносить изменения в имеющиеся инструментальные средства программирования на БЯ. Вместо этого для реализации таких расширений необходимо разработать библиотеку функций, включающую разбор и интерпретацию строк, описывающих конструкции расширений. Разработка выполняется средствами БЯ программирования. В зависимости от структуры обобщенного строкового описания конструкции расширения реализация такой функции может быть как относительно простой, например, основанной на принципе разбора регулярных выражений, так и сложной, требующей реализации более сложного синтаксического анализа. Однако у такого подхода есть и серьезные недостатки [15]:

- конструкции расширений не являются частью языка, а вводятся в программу с помощью вспомогательных конструкций — в виде обрамленной в кавычки строки расширения и явных вызовов функций для ее формирования и интерпретации;

• передача «параметров» между основным кодом на БЯ и кодом расширения не контролируется компилятором;

• отсутствует специализированная типизация возвращаемого значения функции интерпретации расширения, поскольку результат работы этой функции представлен в виде объектов БЯ, а не в виде объектов, соответствующих специфическим понятиям данного расширения (см. пример 6, строки 10–12);

• невозможно осуществить проверки времени компиляции для такого рода расширений.

*Окончание следует.*

## Литература


1. **Карпов Ю. Г.** Теория и технология программирования. Основы построения трансляторов. — СПб.: БХВ-Петербург, 2005. — 272 с.
2. **Dmitriev S.** Language Oriented Programming: The Next Programming Paradigm. — JetBrains, 2004. http://www.onboard.jetbrains.com/is1/articles/04/10/lop/, www.sergeydmitriev.com **(дата обращения: 15.07.2009).**
3. **Horowitz E.** Fundamentals of Programming Languages. Second Ed. — USA: Computer Science Press, 1984. — 446 p.
4. **Davis T. A., Sigmon K.** MATLAB Primer. Seventh Ed. — Chapman & Hall/CRC, 2005. — 215 p.
5. **Михеева В. Д., Скрипниченко В. И.** Расширение языка Object Pascal (Delphi) таблично-ориентированными средствами решения задач эфемеридной астрономии // Сообщения ИПА РАН. — СПб., 2006. № 168. — 20 с.
6. **Михеева В. Д., Новиков Ф. А., Скрипниченко В. И.** Дельта-язык и система программирования для решения прикладных задач с табличными данными // Научно-технические ведомости СПбГПУ. 2007. Т. 2. № 4. С. 57–60.
7. **Krasinsky G. A., Novikov F. A., Skripnichenko V. I.** Problem Oriented Language for Ephemeris Astronomy and its Realization in System ERA // Cel. Mech. 1989. Vol. 45. P. 219–229.
8. **International** standard: ISO/IEC 14882:2003(E): Programming Languages — C++. ISO/IEC JTC1/SC22/WG21: The C++ Standard Committee. http://www.open-std.org/jtc1/sc22/wg21/ (дата обращения: 10.09.2009).
9. **Михеева В. Д., Харитонова И. А.** Microsoft Access 2002 в подлиннике. Наиболее полное руководство. — СПб.: БХВ-Санкт-Петербург, 2003. — 1040 с.
10. **DataSet** Class, C# // MSDN Library for Visual Studio 2005. ms-help://MS.VSCC.v80/MS.MSDN.v80/MS.NETFX30SDK4VS.1033/cpref9/html/T_System_Data_DataSet.htm (дата обращения: 17.08.2009); http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx (дата обращения: 04.11.2009).
11. **C++** Language Reference, Using extern to Specify Linkage (extern modifier, linkage to non-C++ functions) // MSDN Library. Jan. 2006. ms-help://MS.MSDNQTR.2006JAN.1033/vclang/html/_pluslang_Linkage_to_Non.2d.C.2b2b_.Functions.htm **(дата обращения: 17.08.2009).**
12. **Open** SystemC Initiative (OSCI). http://www.systemc.org/home (дата обращения: 15.07.2009).
13. **IEEE 1666-2005** Standard SystemC Language Reference Manual (LRM). http://standards.ieee.org/getieee/1666/index.html (дата обращения: 15.07.2009).
14. **SystemC** Synthesizable Subset. Whitepaper. Draft 1.1.18. December 23, 2004. Synthesis Working Group (SWG) of Open SystemC Initiative (OSCI). http://www.systemc.org/downloads/drafts_review/ **(дата обращения: 14.01.2005).**
15. **Hejlsberg A.** DEV223: The .NET Language Integrated Query (LINQ) Overview // Microsoft Tech Ed Developers, 7–10 Nov. 2006. Barcelona, Spain. http://msmvps.com/blogs/vandooren/archive/2006/11/07/Tech_2D00_Ed-developers-Barcelona_3A00_-Tuesday.aspx (дата обращения: 21.03.2010).