



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Прикладная математика»

Программирование в Delphi: модули

Методические указания к лабораторной работе № 12
по курсам «Информатика», «Алгоритмические языки
и программирование»

Автор
Е.Н. Ладоса, Д.С. Цымбалов, О.В. Яценко,
А.П.Мул, В.И.Снигирева

Ростов-на-Дону, 2018

Аннотация

Описываются принципы и методы модульного программирования в Object Pascal. Целью работы ставится рационализация обработки ПО. Предназначены для студентов всех специальностей факультета «Информатика и вычислительная техника».

Автор

Доцент, к.т.н.
Ладоша Е.Н.

Старший преподаватель кафедры
«Электроника и электротехника»
Цымбалов Д.С.

Доцент, к.ф.-м.н.
Яценко О.В.

Старший преподаватель кафедры
«Прикладная математика»
Мул А.П.

Студент ДГТУ
Снигирева В.И.



Цель работы

Цель работы – изучить принципы и способы модульного программирования в Delphi. Рассмотреть основные вопросы создания, использования и агрегации модулей.

Модули

Модуль — это подключаемая к программе библиотека ресурсов. Он может содержать описания типов, констант, переменных и подпрограмм. В модуль обычно объединяют связанные между собой ресурсы: например, в составе оболочки есть модуль Graph для работы с экраном в графическом режиме.

Модули применяются либо как библиотеки, которые могут использоваться различными программами, либо для разбиения сложной программы на составные части.

Чтобы использовать модуль, достаточно знать только его интерфейс: детали реализации модуля скрыты от его пользователя. Это позволяет успешно создавать программы большого объема, поскольку мозг человека может хранить одновременно довольно ограниченный объем информации. Кроме того, если программа разбита на модули, возрастает скорость ее компиляции, поскольку модули хранятся в готовом, скомпилированном виде и перекомпилируются только при наличии изменений в их исходном тексте.

Задача разбиения программы на максимально обособленные части, спецификации их интерфейсов и оформления этих частей в виде модулей должна решаться на этапе проектирования программы.

Использование модулей имеет еще одно важное преимущество: оно позволяет преодолеть ограничение в один сегмент на объем кода исполняемой программы, поскольку код каждого подключаемого к программе модуля содержится в отдельном сегменте.

Модули можно разделить на стандартные, которые входят в состав системы программирования, и пользовательские, то есть создаваемые программистом. Чтобы подключить модуль к программе, его требуется предварительно скомпилировать. Результат компиляции каждого модуля хранится на диске в отдельном файле с расширением .tpu.

Рассмотрим правила оформления модулей.

Структура модулей

Модуль имеет следующую структуру:

```
Unit <имя>;  
interface  
<Интерфейсная часть>  
implementation
```

```
<исполняемая часть>  
initialization  
<иницилирующая часть>  
finalization  
<завершающая часть>  
end.
```

Здесь **Unit** - зарезервированное слово; начинает заголовок модуля; *<имя>* - имя модуля (правильный идентификатор); **interface** - зарезервированное слово; начинает интерфейсную часть модуля; **implementation** - зарезервированное слово; начинает исполняемую часть; **initialization** - зарезервированное слово; начинает иницилирующую часть модуля; **finalization** - зарезервированное слово; начинает заключительную часть модуля; **end** - зарезервированное слово - признак конца модуля.

Таким образом, модуль состоит из заголовка и четырех составных частей, любая из которых может быть пустой.

Заголовок модуля и связь модулей друг с другом

Заголовок модуля состоит из зарезервированного слова Unit и следующего за ним имени модуля. Для правильной работы среды Pascal и возможности подключения средств, облегчающих разработку крупных программ, это имя должно совпадать с именем дискового файла, в который помещается исходный текст модуля. Если, например, имеется заголовок

```
Unit Global;
```

то исходный текст соответствующего модуля должен размещаться в дисковом файле GLOBAL.PAS. Имя модуля служит для его связи с другими модулями и основной программой. Эта связь устанавливается специальным предложением

```
Uses <сп.модулей>
```

Здесь Uses - зарезервированное слово (использует); <сп.модулей> – список модулей, с которыми устанавливается связь; элементами списка являются имена модулей, отделяемые друг от друга запятыми, например:

```
Uses Windows, SysUtils, MyUnit;
```

Если объявление **Uses** используется, оно должно открывать раздел описаний основной программы. Модули могут использовать другие модули. Предложение **Uses** в модулях может следовать либо сразу за зарезервированным словом interface, либо сразу за словом implementation, либо, наконец, и там, и там (т.е. допускаются два предложения **Uses**).

Интерфейсная часть

Интерфейсная часть открывается зарезервированным словом **interface**.

В этой части содержатся объявления всех глобальных объектов модуля (типов, констант, переменных и подпрограмм), которые должны стать доступными основной программе и/или другим модулям. При объявлении глобальных подпрограмм в интерфейсной части указывается только их заголовок, например:

```
Unit Cmplx;  
Interface  
type  
    Complex = record  
        re,im: Real  
    end;  
function AddC(x,y: Complex): Complex;  
function MulC(x,y: Complex): Complex;
```

Если теперь в другом модуле написать предложение

```
Uses Cmplx;
```

то в нем станут доступными тип `Complex` и две процедуры - `AddC` и `MulC` из модуля `Cmplx`.

Исполняемая часть

Исполняемая часть начинается зарезервированным словом **implementation** и содержит описания подпрограмм, объявленных в интерфейсной части. В ней могут объявляться локальные для модуля объекты – вспомогательные типы, константы, переменные и блоки, а также метки, если они используются в инициализирующей части.

Описанию подпрограммы, объявленной в интерфейсной части модуля, в исполняемой части должен предшествовать заголовок, в котором можно опускать список формальных параметров (и тип результата для функции), так как они уже описаны в интерфейсной части. Но если заголовок подпрограммы приводится в полном виде, т.е. со списком формальных параметров и объявлением результата, он должен совпадать с заголовком, объявленным в интерфейсной части, например:

```
Unit Cmplx;  
Interface  
type  
    Complex = record  
        re,im: real  
    end;  
procedure AddC (x,y: Complex; var z: Complex);  
procedure MulC (x,y: Complex; var z: Complex);
```

```
Implementation
```

```
procedure AddC (x,y: Complex; var z: Complex);  
begin  
...  
end;  
  
procedure MulC; // Вариант описания подпрограммы без  
                // повторения списка параметров  
begin  
...  
end;  
end.
```

Замечу, что хотя и допускается краткое объявление заголовка подпрограммы (как в предыдущем примере - функции MulC), тем не менее использовать такую форму в серьезной программе не рекомендуется: перечень параметров непосредственно в заголовке подпрограммы облегчает чтение кода и понимание деталей реализации алгоритма. Повторение заголовка в исполняемой части должно быть полным и точным. Если бы мы использовали заголовок

```
procedure AddC (x,y: Complex; var z: Complex);  
begin  
...  
end;
```

компилятор немедленно известил бы нас о несовпадении заголовка с объявлением функции в интерфейсной части (второй параметр должен иметь имя y).

Иницилирующая и завершающая части

В иницилирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти операторы выполняются до передачи управления основной программе и обычно используются для подготовки ее работы. Например, в них могут инициализироваться переменные, открываться нужные файлы и т.д. Операторы завершающей части выполняются в момент окончания работы программы.

Иницилирующая и завершающая части модуля используются крайне редко. Если они есть, то при старте программы выполняются иницилирующие части всех модулей в порядке их перечисления в заголовке программы (в предложении **Uses** файла проекта). В том же порядке выполняются завершающие части при окончании работы программы.

Доступ к объявленным в модуле объектам

Пусть, например, мы создаем модуль, реализующий арифметику комплексных чисел (такая арифметика ни в стандартном Паскале, ни в Object Pascal

не предусмотрена). Арифметика комплексных чисел реализуется четырьмя функциями:

```
nit Cmplx;  
Interface  
type  
    Complex = record  
        re,im: real  
    end;  
procedure AddC (x,y: Complex; var z: Complex);  
procedure SubC (x,y: Complex; var z: Complex);  
procedure MulC (x,y: Complex; var z: Complex);  
procedure DivC (x,y: Complex; var z: Complex);
```

```
const  
    c : Complex = (re : 0.1; im : -1);
```

Implementation

```
procedure AddC (x,y: Complex; var z: Complex);
```

```
begin  
    z.re := x.re + y.re;  
    z.im := x.im + y.im  
end;
```

```
procedure SubC (x,y: Complex; var z: Complex);  
begin  
    z.re := x.re - y.re;  
    z.im := x.im - y.im  
end;
```

```
procedure MulC (x,y: Complex; var z: Complex);  
begin  
    z.re := x.re * y.re - x.im * y.im;  
    z.im := x.re * y.im + x.im * y.re  
end;
```

```
procedure DivC (x,y: Complex; var z: Complex);  
var
```



```
f: Real;  
begin  
  f := sqr(y.re) + sqr(y.im);  
  if f=0 then begin  
    z.re := (x.re * y.re + x.im * y.im) / f;  
    z.im := (x.re * y.im - x.im * y.re) / f;  
  end  
  else begin  
    z.re := 1.7e38;  
    z.im := 1.7e38;  
  end  
end;  
end.
```

Задачи

1. Для отладки модуля, реализующего арифметику комплексных чисел, разработайте тестовую программу.
2. Разработайте *пакет подпрограмм матричной алгебры*. Модуль кода MatrixUnit должен содержать подпрограммы, выполняющие операции суммирования (procedure Add(matrix1, matrix2: MatrixType; var matrix: MatrixType;), умножения (procedure Multiply(matrix1, matrix2: MatrixType; var matrix: MatrixType;) и скалярного умножения матриц (procedure ScalarMultiply(matrix: MatrixType; multiple: Real; var matrix: MatrixType;) и вспомогательную процедуру заполнения **результатирующей** матрицы нулями (procedure Zero(var matrix: MatrixType;), где MatrixType запись с полями rows: Integer; cols: Integer; grid: array [1..n, 1..m] of Real; . Для отладки пакета разработайте тестовую программу.

Список использованной литературы

1. Фаронов В.В. Delphi 3. Учебный курс. М.: «Нолидж», 1998. 400 с.
2. Галисеев Г.В. Программирование в среде Delphi 8 for .NET. М.: Издательский дом «Вильямс», 2004. 304 с.
3. Павловска Т.А. Паскаль. Программирование на языке высокого уровня. СПб.: Питер, 2003. 393 с.
4. Абрамов С.А. и др. Задачи по программированию. М.: Наука, 1988. 224 с.

