**Содержание**

Введение

1. Теоретическая часть. Управляющие структуры языка Delphi

2. Анализ задачи

3. Блок – схема алгоритма

4. Руководство пользователя

4.1 Системные требования

4.2 Структура программы

4.3 Инструкция пользователя

5. Тестовый пример

Заключение

Список использованной литературы

Приложение

# Введение

Delphi – среда визуального программирования. Визуальное программирование базируется на объектно-ориентированной технологии. Структурной единицей визуального программирования, основным «строительным элементом» для программы является компонент.

Компонент – это разновидность класса, который представлен пиктограммой на палитре компонентов Delphi, может быть визуально перенесен в программу и имеет набор свойств, которые можно определять, не изменяя текста программ. В этом суть визуального программирования.

Как и любой класс, компонент характеризуется полями, свойствами и методами. В Delphi вместо полей обычно используются свойства. Свойство можно рассматривать как разновидность поля класса, обращение к которому автоматически приводит к вызову метода чтения / записи поля. (Напомним, что хорошим стилем в объектно-ориентированном программировании считается обращение к полям только через методы).

Кроме того, компонент имеет перечень событий, на которые он способен реагировать (например, нажатие клавиши, щелчок кнопкой мыши и др.). Задача программиста – написать обработчики событий, отвечающие за реакцию компонента на определенное событие.

Компоненты бывают визуальными и невизуальными. Первые предназначены для организации интерфейса с пользователем (кнопки, строки редактирования, переключатели, списки и т.д.). Они видны на экране во время выполнения программы. Невизуальные компоненты служат для доступа к системным ресурсам, например, драйверам баз данных. Во время работы приложения они, как правило, не видны.

# 1. Теоретическая часть. Управляющие структуры языка Delphi

На практике редко встречаются задачи, алгоритм решения которых является линейным. Часто оказывается, что алгоритм решения даже элементарной задачи не является линейным. Например, пусть надо вычислить по формуле ток в электрической цепи. Если предположить, что пользователь всегда будет вводить верные данные, то алгоритм решения этой задачи действительно является линейным. Однако полагаться на то, что пользователь будет вести себя так, как надо программе, не следует. Формула расчета предполагает, что величина сопротивления не равна нулю. А что будет, если пользователь введет 0? Ответ простой: возникнет ошибка «Деление на ноль», и программа аварийно завершит работу. Можно, конечно, возложить ответственность за это на пользователя, но лучше внести изменения в алгоритм решения (рис. 1), чтобы расчет выполнялся только в том случае, если введены верные данные.

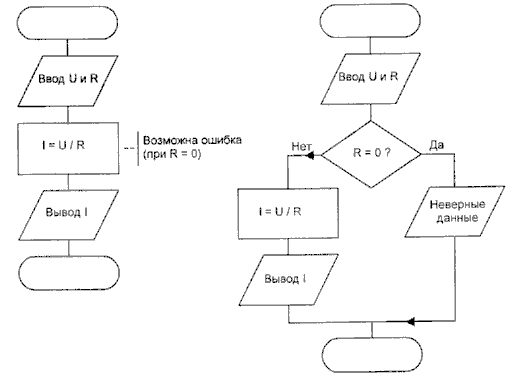


Рис. 1. Два варианта алгоритма решения одной задачи

Точки алгоритма, в которых выполняется выбор дальнейшего хода программы, называются точками выбора. Выбор очередного шага решения задачи осуществляется в зависимости от выполнения некоторого условия.

Условие

В повседневной жизни условие обычно формулируется в виде вопроса, на который можно ответить Да или Нет. Например:

Величина сопротивления равна нулю?

Ответ правильный?

Сумма покупки больше 300 рублей?

В программе условие – это выражение логического типа (Boolean), которое может принимать одно из двух значений: True (истина) или False (ложь).

Простое условие состоит из двух операндов и оператора сравнения. В общем виде условие записывается следующим образом: Оn1 Оператор On2

где:

On1 и Оп2 – операнды условия, в качестве которых может выступать переменная, константа, функция или выражение;

Оператор – оператор сравнения.

В языке Delphi есть шесть операторов сравнения, которые приведены в табл. 1.

Таблица 2.1. Операторы сравнения

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Результат сравнения |
| > | Больше | True, если первый операнд больше второго, иначе False |
| < | Меньше | True, если первый операнд меньше второго, иначе False |
| = | Равно | True, если первый операнд равен второму, иначе False |
| <> | Не равно | True, если первый операнд не равен второму, иначе False |
| >= | Больше или равно | True, если первый операнд больше или равен второму, иначе False |
| <= | Меньше или равно | True, если первый операнд меньше или равен второму, иначе False |

Ниже приведены примеры условий: Summa < 1000 Score >= HBound Sim = Chr(13).

В первом примере операндами условия является переменная и константа. Значение этого условия зависит от значения переменной Summa. Условие будет верным и, следовательно, иметь значение True, если значение переменной Summa меньше, чем 1000. Если значение переменной Summa больше или равно 1000, то значение этого условия будет False.

Во втором примере в качестве операндов используются переменные. Значение этого условия будет True, если значение переменной Score больше или равно значению переменной HBound.

В третьем примере в качестве второго операнда используется функция. Значение этого условия будет True, если в переменной Sim находится символьный код клавиши <Enter>, равный 13.

При записи условий следует обратить особое внимание на то, что операнды условия должны быть одного типа или, если тип операндов разный, то тип одного из операндов может быть приведен к типу другого операнда. Например, если переменная Key объявлена как integer, то условие Key = Chr(13) синтаксически неверное, т.к. значение возвращаемое функцией Chr имеет тип char (символьный).

Во время трансляции программы при обнаружении неверного условия компилятор выводит сообщение: incompatible types (несовместимые типы).

Из простых условий при помощи логических операторов: and – «логическое И», or – «логическое ИЛИ» и not – «отрицание» можно строить сложные условия.

В общем виде сложное условие записывается следующим образом: условие1 оператор условие2

где:

условие1 и условие2 – простые условия (выражения логического типа);

оператор – оператор and или or.

Например:

(ch >= '0') and (ch <= '9')

(day = 7) or (day = 6)

(Forml. Editl. Text <> ' ') or (Forml. Edit2. Text <> «)

Forml. CheckBoxl. Checked and (Forml. Editl. Text <> «)

Результат выполнения логических операторов and, or и not представлен в табл. 2.

Таблица 2. Выполнение логических операций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Op1 | Op2 | Opt and Op2 | Op1 or Op2 | not Op1 |
| False | False | False | False | True |
| False | True | False | True | True |
| True | False | False | True | False |
| True | True | True | True | False |

При записи сложных условий важно учитывать то, что логические операторы имеют более высокий приоритет, чем операторы сравнения, и поэтому простые условия следует заключать в скобки.

Например, пусть условие предоставления скидки сформулировано следующим образом: «Скидка предоставляется, если сумма покупки превышает 100 руб. и день покупки – воскресенье», Если день недели обозначен как переменная Day целого типа, и равенство ее значения семи соответствует воскресенью, то условие предоставления скидки можно записать: (Summa > 100) and (Day = 7).

Если условие предоставления скидки дополнить тем, что скидка предоставляется в любой день, если сумма покупки превышает 500 руб., то условие можно записать: ((Summa > 100) and (Day =7)) or (Summa > 500).

Выбор

Выбор в точке разветвления алгоритма очередного шага программы может быть реализован при помощи инструкций if и case. Инструкция if позволяет выбрать один из двух возможных вариантов, инструкция case – один из нескольких.

Инструкция if

Инструкция if позволяет выбрать один из двух возможных вариантов развития программы. Выбор осуществляется в зависимости от выполнения условия.

В общем виде инструкция if записывается так:

if условие then

begin

 // здесь инструкции, которые надо выполнить,

 // если условие истинно.

end

else

begin

 // здесь инструкции, которые надо выполнить, // если условие ложно. end;

Обратите внимание, что перед else (после end) точка с запятой не ставится.

Выполняется инструкция if следующим образом:

1. Вычисляется значение условия (условие – выражение логического типа, значение которого может быть равно True или False).

2. Если условие истинно (значение выражения условие равно True), то выполняются инструкции, следующие за словом then (между begin и end). На этом выполнение операции if заканчивается, то есть инструкции, следующие за else, не будут выполнены.

Если условие ложно (значение выражения условие равно False), то выполняются инструкции, следующие за словом else (между begin и end).

На рис. 2 представлен алгоритм, соответствующий инструкции if–tnen–else.

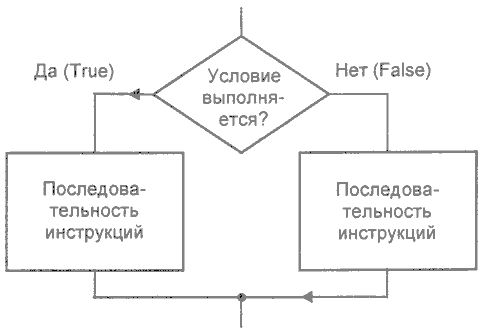


Рис. 2. Алгоритм, реализуемый инструкцией if–then–else

Например, если переменная t обозначает тип соединения сопротивлений в электрической цепи (t=1 соответствует последовательному соединению, t=2 – параллельному), a r1 и r2 – величины сопротивлений, то приведенная ниже инструкция if осуществляет выбор формулы, по которой будет выполнен расчет.

if t=l then

begin

z:=r1+r2;

end

else

begin

z:=(r1+r2)/(r1\*r2);

end;

Если в инструкции if между begin и end находится только одна инструкция, то слова begin и end можно не писать.

Например, инструкцию

if otv=3

then

begin

prav:=prav+1;

end

else

begin

ShowMessage ('Ошибка!');

end;

можно переписать так:

if otv=3 then

prav:=prav+l

else

ShowMessage ('Ошибка!');

Если какое–либо действие должно быть выполнено только при выполнении определенного условия и пропущено, если это условие не выполняется, то инструкция if может быть записана так:

if условие then

begin

{инструкции, которые надо выполнить, если условие выполняется, истинно} end

На рис. 3 представлен алгоритм, соответствующий инструкции if–then.

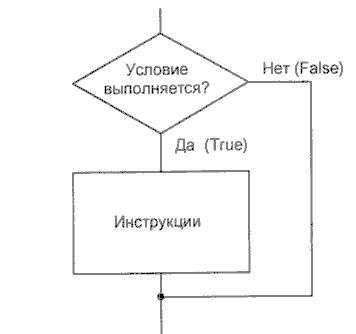


Рис. 3. Алгоритм, реализуемый инструкцией if–then

Например, инструкция

if n=m

then c:=c+l;

увеличивает значение переменной с только в том случае, если значения переменных n и m равны.

Инструкция case

В предыдущем примере, в программе контроля веса, множественный выбор был реализован при помощи вложенных одна в другую инструкций if. Такой подход не всегда удобен, особенно в том случае, если количество вариантов хода программы велико.

В языке Delphi есть инструкция case, которая позволяет эффективно реализовать множественный выбор. В общем виде она записывается следующим образом:

case Селектор of список1:

begin

{инструкции 1} end; список2:

begin

{инструкции 2} end; списокМ:

begin

{инструкции N}

end;

else

begin

(инструкции)

end;

end;

где:

Селектор – выражение, значение которого определяет дальнейший ход выполнения программы (т.е. последовательность инструкций, которая будет выполнена);

Список N – список констант. Если константы представляют собой диапазон чисел, то вместо списка можно указать первую и последнюю константу диапазона, разделив их двумя точками. Например, список 1, 2, 3, 4, 5, 6 может быть заменен диапазоном 1..6.

Выполняется инструкция case следующим образом:

1. Сначала вычисляется значение выражения–селектора.

2. Значение выражения–селектора последовательно сравнивается с константами из списков констант.

3. Если значение выражения совпадает с константой из списка, то выполняется соответствующая этому списку группа инструкций. На этом выполнение инструкции саsе завершается.

4. Если значение выражения–селектора не совпадает ни с одной константой из всех списков, то выполняется последовательность инструкций, следующая за else.

Синтаксис инструкции case позволяет не писать else и соответствующую последовательность инструкций. В этом случае, если значение выражения не совпадает ни с одной константой из всех списков, то выполняется следующая за case инструкция программы.

На рис. 4 приведен алгоритм, реализуемый инструкцией case.

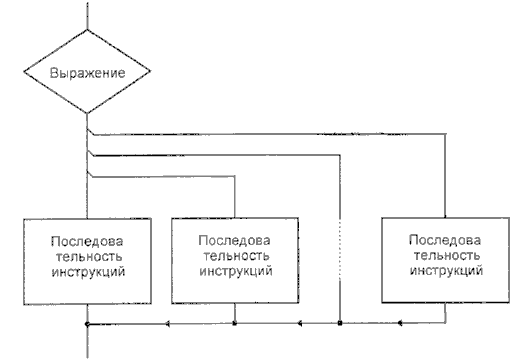


Рис. 4. Алгоритм, реализуемый инструкцией case

Циклы

Алгоритмы решения многих задач являются циклическими, т.е. для достижения результата определенная последовательность действий должна быть выполнена несколько раз.

Например, программа контроля знаний выводит вопрос, принимает ответ, добавляет оценку за ответ к сумме баллов, затем повторяет это действие еще и еще раз, и так до тех пор, пока испытуемый не ответит на все вопросы.

Другой пример. Для того чтобы найти фамилию человека в списке, надо проверить первую фамилию списка, затем вторую, третью и т.д. до тех пор, пока не будет найдена нужная фамилия или не будет достигнут конец списка.

Алгоритм, в котором есть последовательность операций (группа инструкций), которая должна быть выполнена несколько раз, называется циклическим, а сама последовательность операций именуется циклом.

В программе цикл может быть реализован при помощи инструкций for, while и repeat.

Инструкция for

Рассмотрим следующую задачу. Пусть нужно написать программу, которая вычисляет значение функции у = 5х2 – 7 в точках –1, –0.5, 0, 0.5 и 1

(таблица должна быть выведена в поле метки формы приложения). Процедура, обеспечивающая решение поставленной задачи, может выглядеть так:

Процедура TForm1. ButtonlClick (Sender: TObject); var

у: real; // значение функции

x: real; // аргумент функции

dx: real; // приращение аргумента

st: string; // изображение таблицы

begin

st:='';

x:= –1; dx:= 0.5;

у:= 5\*х\*х –7;

st:= st+ FloatToStr(x)+' '+ FloatToStr(y)+chr(13);

x:=x + dx;

у:= 5\*х\*х –7;

st:= st+ FloatToStr(x)+\* '+ FloatToStr(y)+chr(13);

x:=x + dx;

у:= 5\*х\*х –7;

st:= st+ FloatToStr(x)+\* '+ FloatToStr(y)+chr(13);

x:=x + dx;

у:= 5\*х\*х –7;

st:= st+ FloatToStr(x)+' ' + FloatToStr(y)+chr(13);

x:=x + dx;

у:= 5\*х\*х –7;

st:= st+ FloatToStr(x)+' '+ FloatToStr(y)+chr(13);

x:=x + dx;

Label1. Caption:= st;

end;

Из текста процедуры видно, что группа инструкций

у:= 5\*х\*х –7;

st:= st+ FloatToStr(x)+' '+ FloatToStr(y)+chr(13);

x:=x + dx;

Обеспечивающая вычисление значения функции, формирование строки таблицы и увеличение аргумента, выполняется 5 раз.

Воспользовавшись инструкцией for, приведенную процедуру можно переписать следующим образом:

Процедура TForm1. ButtonlClick (Sender: TObject);

var

у: real; // значение функции

x: real; // аргумент функции

dx: real; // приращение аргумента

st: string; // изображение таблицы

i: integer; // счетчик циклов

begin

st:=''; x:= –1; dx:= 0.5;

for i:=l to 5 do begin

у:= 5\*x\*x –7;

st:= st+ FloatToStr(x)+' '+ FloatToStr(y)+chr(13); x:=x + dx; end;

Label1. Caption:= st;

end;

Второй вариант процедуры, во-первых, требует меньше усилий при наборе, во-вторых, процедура более гибкая: для того чтобы увеличить количество строк в выводимой таблице, например до десяти, достаточно в строке for i:=1 to 5 do число 5 заменить на 10.

Инструкция for используется в том случае, если некоторую последовательность действий (инструкций программы) надо выполнить несколько раз, причем число повторений заранее известно.

В общем виде инструкция for записывается следующим образом: for счетчик:= нач\_знач to кон\_знач do begin

 // здесь инструкции, которые надо выполнить несколько раз end

где:

счетчик – переменная–счетчик числа повторений инструкций цикла;

нач\_знач – выражение, определяющее начальное значение счетчика циклов;

кон\_знач – выражение, определяющее конечное значение счетчика циклов.

Переменная счетчик, выражения нач\_знач и кон\_знач должны быть целого типа.

Количество повторений инструкций цикла можно вычислить по формуле

(кон\_знач – нач\_знач + l).

Примеры:

for i:=l to 10 do begin

label1.caption:=label1.caption + '\*'; end;

for i: =1 to n do s:= s+i;

Примечание

Если между begin и end находится только одна инструкция, то слова begin и end можно не писать.

Алгоритм, соответствующий инструкции for, представлен на рис. 5. Обратите внимание, что если начальное значение счетчика больше конечного значения, то последовательность операторов между begin и end не будет выполнена ни разу.

Кроме того, после каждого выполнения инструкций тела цикла счетчик циклов увеличивается автоматически.

Переменную–счетчик можно использовать внутри цикла (но ни в коем случае не изменять).

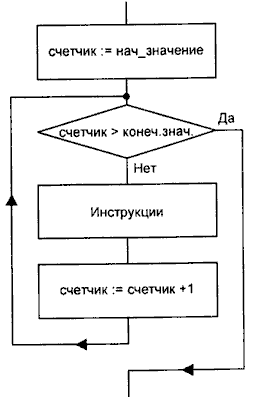


Рис. 5. Алгоритм инструкции for

Инструкция while

Инструкция (цикл) while используется в том случае, если некоторую последовательность действий (инструкций программы) надо выполнить несколько раз, причем необходимое число повторений во время разработки программы неизвестно и может быть определено только во время работы программы.

Типичными примерами использования цикла while являются вычисления с заданной точностью, поиск в массиве или в файле.

В общем виде инструкция while записывается следующим образом:

while условие do begin

 // здесь инструкции, которые надо выполнить несколько раз

end

где условие – выражение логического типа, определяющее условие выполнения инструкций цикла.

1. Инструкция while выполняется следующим образом:

2. Сначала вычисляется значение выражения условие.

3. Если значение выражения условие равно False (условие не выполняется), то на этом выполнение инструкции while завершается.

4. Если значение выражения условие равно True (условие выполняется), то выполняются расположенные между begin и end инструкции тела цикла. После этого снова проверяется выполнение условия. Если условие выполняется, то инструкции цикла выполняются еще раз. И так до тех пор, пока условие не станет ложным (False).

Алгоритм, соответствующий инструкции while, представлен на рис. 6.

Внимание!

Для того чтобы инструкции цикла while, которые находятся между begin и end, были выполнены хотя бы один раз, необходимо, чтобы перед выполнением инструкции while значение выражения условие было истинно.

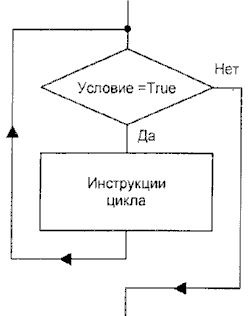


Рис. 6. Алгоритм инструкции while

Для того чтобы цикл завершился, нужно, чтобы последовательность инструкций между begin и end влияла на значение выражения условие (изменяла значения переменных, входящих в выражение условие).

Инструкция repeat

Инструкция repeat, как и инструкция while, используется в программе в том случае, если необходимо выполнить повторные вычисления (организовать цикл), но число повторений во время разработки программы неизвестно и может быть определено только во время работы программы, т.е. определяется ходом вычислений.

В общем виде инструкция repeat записывается следующим образом:

repeat

 // инструкции unti1 условие

где условие – выражение логического типа, определяющее условие завершения цикла.

Инструкция repeat выполняется следующим образом:

1. Сначала выполняются находящиеся между repeat и until инструкции тела цикла.

2. Затем вычисляется значение выражения условие. Если условие ложно (значение выражения условие равно False), то инструкции тела цикла выполняются еще раз.

3. Если условие истинно (значение выражения условие равно True), то выполнение цикла прекращается.

Таким образом, инструкции цикла, находящиеся между repeat и unti1, выполняются до тех пор, пока условие ложно (значение выражения условие  
равно False).

Алгоритм, соответствующий инструкции repeat, представлен на рис. 7.

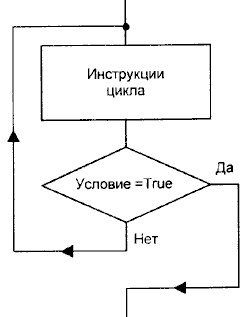


Рис. 7. Алгоритм, соответствующий инструкции repeat

Внимание!

Инструкции цикла, находящиеся между repeat и until, выполняются как минимум один раз. Для того чтобы цикл завершился, необходимо, чтобы инструкции цикла, располагающиеся между repeat и until, изменяли значения переменных, входящих в выражение условие.

Инструкция goto

Инструкции if и case используются для перехода к последовательности инструкций программы в зависимости от некоторого условия. Поэтому их иногда называют инструкциями условного перехода. Помимо этих инструкций управления ходом выполнения программы существует еще одна – инструкция безусловного перехода goto.

В общем виде инструкция goto записывается следующим образом:

goto Метка

где метка – это идентификатор, находящийся перед инструкцией, которая должна быть выполнена после инструкции goto.

Метка, используемая в инструкции goto, должна быть объявлена в разделе меток, который начинается словом label и располагается перед разделом объявления переменных.

В программе метка ставится перед инструкцией, к которой должен быть выполнен переход в результате выполнения инструкции goto. Сразу после метки ставится двоеточие.

В литературе по программированию можно встретить суждения о недопустимости применения инструкции goto, поскольку она приводит к запутанности программ. Однако с категоричностью таких утверждений согласиться нельзя. В некоторых случаях применение инструкции goto вполне оправдано.

# 2. Анализ задачи

Для решения задачи необходимы следующие исходные данные:

* число строк матрицы – целочисленное значение;
* число столбцов матрицы – целочисленное значение;
* значения элементов матрицы – целочисленные значения.

Примем ограничение на размерность матрицы – число строк и столбцов не должно превышать 10.

Пошаговый алгоритм решения задачи.

Исходные данные:

n, m – число строк и столбцов соответственно;

aij – элементы матрицы;

Сij – матрица цветов каждого элемента;

с – начальный цвет слоя;

di – массив, составленный из элементов слоев;

k – число элементов в слое;

fn – индекс слоя (номер строки и столбца с которого он начинается), начальное значение 1.

1) Формирование слоев.

1. Помещаем в d элементы fn – й строки;
2. Помещаем в d элементы m – го столбца начиная с (fn + 1) – й строки по n – ю строку;
3. Помещаем в d элементы n – й строки, начиная с m – 1 столбца по fn столбец;
4. Помещаем в d элементы fn – го столбца начиная с n – 1‑й строки по fn + 1 строку;

В соответствующие элементы матрицы цветов С помещаем значение с.

2) Сдвигаем элементы массива d на 1 позицию вправо.

1. запоминаем значение последнего элемента t = dk;
2. выполняем присвоение di = di–1 для i = k, 2;
3. помещаем последний элемент на место первого d1 = t.

3) Изменяем значения:

fn = fn + 1,

n = n – 1,

m = m – 1

и повторяем шаги алгоритма с 1 по 3, пока n > 2 и m > 2.

# 3. Блок – схема алгоритма

Начало

Ввод n, m, a

c 🡨 clBlack

k = 1

n ≥ 2 или m ≥ 2 и fn < n и fn < m

j = 0

i = fn, m

c[fn, i] = cc[k]

j = j + 1, d[j] = a[fn, i]

i = fn + 1, n,

c[i, m] = cc[k]

j = j + 1, d[j] = a[i, m]

i = m – 1, fn

c[n, i] = cc[k]

j = j + 1, d[j] = a[n, i]

2

1

3

да

нет

2

1

3

i = n – 1, fn + 1

c[i, fn] = cc[k]

j = j + 1, d[j] = a[i, fn]

Сдвиг слоя Sdvig(d, j)

j = 0

i = fn, m,

j = j + 1

a[fn, i] = d[j]

i = fn + 1, n

j = j + 1

a[i, m] = d[j]

i = m – 1, fn

j = j + 1

a[n, i] = d[j]

i = n – 1, fn + 1

j = j + 1

a[i, fn] = d[j]

4

1

3

4

1

3

k=k+1

Рекурсивный вызов SloiColor(fn + 1, n – 1, m – 1, sd)

Конец

# 4. Руководство пользователя

## 

## 4.1 Системные требования

Для функционирования приложения необходим персональный компьютер со следующими характеристиками:

* процессор CPU – не хуже Pentium;
* оперативная память RAM – не менее 16 Мб;
* свободное пространство на жестком диске – не менее 500 Кбайт;
* стандартная мышь для COM порта;
* стандартная IBM PC/АT клавиатура;

Приложение работает под управлением операционной системы Windows 95/98/2000/ХР. Наличие среды визуального программирования для работы программы не является обязательным.

## 

## 4.2 Структура программы

Программа состоит из следующих программных модулей:

unit1 – главный интерфейсный модуль;

ms – проект.

Рассмотрим состав главного модуля.

В нем определены два пользовательских типа данных:

mas – одномерный массив целых чисел из 100 элементов;

mat – двумерный массив целых чисел размерности 10×10 элементов.

Цвета слоев определяет константа – массив элементов TColor – cc.

Наиболее значимые переменные программы.

mf, тип string – имя файла с матрицей;

c – двумерный массив цветов слоев с элементами типа TColor;

a, тип mat – заданная матрица;

k, тип integer – счетчик слоев;

d, тип mas – одномерный массив с элементами слоя;

n, m, тип integer – число строк и столбцов матрицы;

f, тип textfile – текстовый файл из которого считывается матрица.

Процедуры пользователя.

Sdvig (d, n)

Назначение: сдвиг массива по часовой стрелке.

Параметры:

d, тип mas – одномерный массив с элементами слоя;

n, тип integer – число элементов массива;

SloiColor (fn, n, m, sd)

Назначение: определение цветов слоев и их сдвиг.

Параметры:

fn, тип integer – номер определяемого слоя;

n, m, тип integer – число строк и столбцов матрицы;

sd, тип boolean – логическая переменная, имеющая значение True, если производится определение цветов и сдвиг элементов слоя и False – если только определяется цвет.

MQuitClick

Назначение: выход из программы.

MOpenFileClick

Назначение: открыть файл данных.

FillGrid (a, n, m)

Назначение: вывод матрицы с заданными цветами элементов.

Параметры:

a, тип mat – заданная матрица;

n, m, тип integer – число строк и столбцов матрицы;

MRunClick

Назначение: управляющая процедура сдвига слоев матрицы.

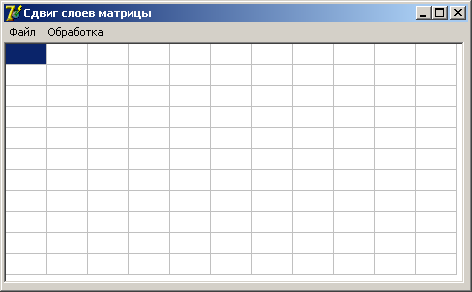


Рис. 8. Окно программы

## 4.3 Инструкция пользователя

Рассмотрим последовательность действий пользователя при использовании программы для сдвига элементов матрицы. Ему необходимо запустить на выполнение программу, которая называется «ms.exe».

Для удобства запуска программы под управлением Microsoft Windows, можно сделать следующее: щелкните правой кнопкой мыши по Рабочему Столу Windows ⇒ в появившемся меню выберите Создать ⇒ Ярлык ⇒ в появившемся окне в Командной строке набираем путь доступа к программе и нажимаем кнопку Далее ⇒ в следующем окне назовем наш ярлык Сдвиг и нажимаем кнопку Далее ⇒ выберите иконку и нажмите кнопку Готово. Теперь у Вас на рабочем столе находится иконка программы.

После запуска пользователь увидит на экране окно, изображенное на рисунке 8.

В верхней строке окна находится главное меню с двумя пунктами:

Файл. Содержит подпункты:

* Открыть – загрузка матрицы из текстового файла;
* Выход – закончить работу с программой.

Обработка, содержит один подпункт Начать – запуск алгоритма сдвига элементов матрицы.

Остальную часть окна занимает область вывода матрицы.

Порядок работы:

* выбрать файл с исходными данными Файл 🡪 Открыть;
* выполнить команду Обработка 🡪 Начать для сдвига элементов слоев.

Для хранения матриц используется текстовый файл со следующей структурой:

1 строка – два целочисленных значения n, m из диапазона 1…10 разделенные пробелом – число строк и столбцов матрицы соответственно;

В последующих n строка расположена сама матрица, m целочисленных значений в строке, разделитель – пробел.

Необходимо помнить, что обработке подлежат матрицы максимальной размерности 10×10 элементов.

# 5. Тестовый пример



# Заключение

Результатом курсового проектирования по дисциплине «Программирование на языках высокого уровня» на тему «Разработка программы для сдвига элементов матрицы по слоям» является разработанная программа для IBM PC совместимых компьютеров, предназначенная для обработки произвольных целочисленных матриц, а также подробное руководство пользователя программой.

В ходе курсового проектирования были выявлены особенности написания программы, разработан алгоритм программы и описан принцип ее работы.

Недостатком программы является ограниченная размерность обрабатываемых матриц.

В теоретической части работы мной были приведены материалы о программирование в среде Delphi.

# Список использованной литературы

1. Бобровский С.И. Delphi 7. Начальный курс: Учебное пособие – М.: Диалог – МИФИ, 2002. -416 с.
2. Гофман В.Э. Delphi 7. – Спб.: BHV‑Петербург, 2003. -488 с.
3. Лишнер Рэй. Delphi. Справочник. – М.: БЕК, 2003. -715 с.
4. Поган А.М., Царенко Ю.А. Программирование в Delphi. Просто как дважды два. – М.: Нолидж, 2004. -557 с.

# Приложение. Листинг программы

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Menus, Grids, StdCtrls;

type

mas = array [1..100] of integer;

mat = array [1..10, 1..10] of integer;

TForm1 = class(TForm)

MainMenu1: TMainMenu;

N1: TMenuItem;

MOpenFile: TMenuItem;

N4: TMenuItem;

MQuit: TMenuItem;

N6: TMenuItem;

MRun: TMenuItem;

OpenDialog1: TOpenDialog;

StringGrid1: TStringGrid;

procedure MQuitClick (Sender: TObject);

procedure MOpenFileClick (Sender: TObject);

procedure MRunClick (Sender: TObject);

procedure FillGrid (a: mat; n, m: integer);

private

{Private declarations}

public

{Public declarations}

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

const

cc: array [1..10] of TColor = (clBlue, clFuchsia, clGreen,

clLime, clMaroon, clNavy, clOlive,

clRed, clTeal, clYellow);

var

mf: string; // имя файла с матрицей

c: array [1..10, 1..10] of TColor; // массив цветов слоев

a: mat; // заданая матрица

k: integer; // счетчик слоев

procedure Sdvig (var d: mas; n: integer);

 // сдвиг массива по часовой стрелке

var

i, l: integer;

begin

if n = 0 then Exit;

l:= d[n];

for i:= n downto 2 do

d[i]:= d [i – 1];

d[1]:= l;

end;

procedure SloiColor (fn, n, m: integer; sd: boolean);

 // определение цветов слоев и их сдвиг

var

i, j: integer;

d: mas;

begin

if ((n >= 2) or (m >= 2)) and ((fn < n) and (fn < m)) then

begin

j:= 0;

for i:= fn to m do

begin

c [fn, i]:= cc[k];

j:= j + 1;

d[j]:= a [fn, i];

end;

for i:= fn + 1 to n do

begin

c [i, m]:= cc[k];

j:= j + 1;

d[j]:= a [i, m];

end;

for i:= m – 1 downto fn do

begin

c [n, i]:= cc[k];

j:= j + 1;

d[j]:= a [n, i];

end;

for i:= n – 1 downto fn + 1 do

begin

c [i, fn]:= cc[k];

j:= j + 1;

d[j]:= a [i, fn];

end;

if sd then

begin

Sdvig (d, j);

j:= 0;

for i:= fn to m do

begin

j:= j + 1;

a [fn, i]:= d[j];

end;

for i:= fn + 1 to n do

begin

j:= j + 1;

a [i, m]:= d[j];

end;

for i:= m – 1 downto fn do

begin

j:= j + 1;

a [n, i]:= d[j];

end;

for i:= n – 1 downto fn + 1 do

begin

j:= j + 1;

a [i, fn]:= d[j];

end;

end;

Inc(k);

SloiColor (fn + 1, n – 1, m – 1, sd);

end;

end;

procedure TForm1.MQuitClick (Sender: TObject);

 // выход из программы

var

fi: integer;

begin

fi:= Application. MessageBox ('Вы действительно хотите завершить программу?',

'Информация', mb\_YesNo); // вывод окна

if fi = IDYes then

Application. Terminate; // выход

end;

procedure TForm1.MOpenFileClick (Sender: TObject);

 // открыть файл данных

var

n, m, i, j: integer;

f: textfile;

begin

if OpenDialog1. Execute then

begin

mf:= OpenDialog1. FileName;

MRun. Enabled:= True;

AssignFile (f, mf); Reset(f);

ReadLn (f, n, m);

for i:= 1 to n do

begin

for j:= 1 to m do

Read (f, a [i, j]);

ReadLn(f);

end;

CloseFile(f);

for i:= 1 to n do

for j:= 1 to m do

c [i, j]:= clBlack;

k:= 1;

SloiColor (1, n, m, false);

FillGrid (a, n, m);

end;

end;

procedure TForm1. FillGrid (a: mat; n, m: integer);

 // вывод матрицы с заданными цветами элементов

var

i, j: integer;

r: trect;

begin

StringGrid1. Refresh;

for i:= 1 to 10 do

StringGrid1. Cells [i, 0]:= «;

for i:= 1 to 10 do

begin

StringGrid1. Cells [0, i]:= «;

for j:= 1 to 10 do

begin

r:= StringGrid1. CellRect (j, i);

StringGrid1. Canvas. TextOut (r.topleft.x + 1, r. Top + 2, ' ');

end;

end;

for i:= 1 to m do

StringGrid1. Cells [i, 0]:= IntToStr(i);

for i:= 1 to n do

begin

StringGrid1. Cells [0, i]:= IntToStr(i);

for j:= 1 to m do

begin

StringGrid1. Canvas. Font. Color:= c [i, j];

r:= StringGrid1. CellRect (j, i);

StringGrid1. Canvas. TextOut (r.topleft.x + 1, r. Top + 2, IntToStr (a[i, j]));

end;

end;

StringGrid1. Canvas. Refresh;

end;

procedure TForm1.MRunClick (Sender: TObject);

var

i, j, n, m: integer;

f: textfile;

begin

AssignFile (f, mf); Reset(f);

ReadLn (f, n, m);

CloseFile(f);

for i:= 1 to n do

for j:= 1 to m do

c [i, j]:= clBlack;

k:= 1;

SloiColor (1, n, m, true);

FillGrid (a, n, m);

Append(f);

WriteLn(f);

for i:= 1 to n do

begin

for j:= 1 to m do

Write (f, a [i, j]:5);

WriteLn(f);

end;

CloseFile(f);

end;

end.