ДОКУМЕНТАЦИЯ
ПО ИСПОЛЬЗОВАНИЮ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТКИ
FREE OBERON
И ЯЗЫКУ ПРОГРАММИРОВАНИЯ
ОБЕРОН

с исправлениями от 18 июня 2019 г.
для версии Free Oberon 1.0.3

ИЗДАТЕЛЬСТВО
ТЕХНИКО-ТЕОРЕТИЧЕСКОЙ ЛИТЕРАТУРЫ
КОМАНДЫ РАЗРАБОТЧИКОВ FREE OBERON

Р И Г А 2 0 1 9

Оглавление

1. Установка....3
1.1. Установка под ОС GNU/Linux....3
1.2. Установка под ОС Windows....3
2. Проверка работы системы....4
3. Написание программы....4
4. Сохранение файла....5
5. Запуск программы....5
6. Открытие файла....5
7. Перемещение по тексту....5
8. Копирование строк....6
9. Перемещение между открытыми окнами....6
10. Простейшая программа....7
11. Основные типы данных....8
12. Консольный ввод и вывод. Модули In, Out....8
13. Модуль Math....9
14. Модуль Graph....9
14.1. Рисование отрезков....10
14.2. Задание цвета....10
14.3. Получение размера экрана....11
14.4. Манипулирование цветом отдельных пикселей....11
14.5. Очистка экрана....11
14.6. Рисование некоторых фигур....12
14.7. Настройка графического окна....12
14.8. Анимация....12
14.9. Генератор псевдослучайных последовательностей чисел....13
14.10. Работа с изображениями....13
Приложение А. Пересборка Free Oberon под ОС Windows....14

1. Установка.

Free Oberon — кроссплатформенная среда разработки. Она доступна для ОС Windows в виде установщика в формате EXE, а также для ОС GNU/Linux и прочих UNIX-подобных систем в виде архива tar.gz с исходными кодами, которые необходимо скомпилировать, предварительно снабдив операционную систему некоторыми необходимыми библиотеками.

1.1. Установка под ОС GNU/Linux.

1. Скачайте исходные коды Free Oberon в виде архива tar.gz с сайта `freeoberon.su` или с репозитория на GitHub. Архив с версией для ОС Windows тоже подойдёт, потому что он содержит исходные коды. Распакуйте архив в свой домашний каталог или в другое место на диске. (Ниже предполагается, что архив распакован в домашний каталог.)

2. Используя терминал или каким-либо иным образом, установите следующие пакеты программного обеспечения:
   ```
   libsdl2-dev             libsdl2-image-dev
   binutils                gcc
   make
   ```

   Названия пакетов даны в соответствии с их наименованием в ОС Debian GNU/Linux. Они подойдут для пользователей Ubuntu, Linux Mint, Raspbian и др. Для их установки, выполните команду:
   ```
   apt-get install -y libsdl2-dev libsdl2-image-dev
                      binutils gcc make
   ```

   (Эту команду необходимо выполнить с правами суперпользователя, т. е. предварительно необходимо запустить «`su`» и ввести пароль.) На ОС Fedora, Red Hat, CentOS и других, команда и названия пакетов будут отличаться (возможно, также необходимо установить один из двух пакетов: `glibc-static` или `glibc-devel-static`):
   ```
   sudo yum install SDL2-devel SDL2_image-devel
        glibc-devel-static binutils gcc make      (не проверено!)
   ```

3. Войдите в подкаталог «`src`» и запустите компиляцию:
   ```
   cd ~/FreeOberon/src
   make -f Makefile_linux
   ```

4. (по желанию) Допишите в конец файла «`~/.bashrc`» строчку:
   ```
   alias fo='cd ~/FreeOberon;./FreeOberon'
   ```

   чтобы Free Oberon можно было запускать командой «`fo`».

1.2. Установка под ОС Windows.

Скачайте установщик в формате EXE с сайта `freeoberon.su`, запустите его и следуйте инструкциям.

Кроме того, можно скачать версию Free Oberon для Windows в ZIP-архиве и распаковать его в любое место на диске, после чего создать ярлык на рабочем столе.

Примечание. Если вы хотите самостоятельно собрать версию Free Oberon под ОС Windows из исходных кодов, обратитесь к приложению А в данном документе.

2. Проверка работы системы.

Запустите Free Oberon, нажмите `F3` («`Файл → Открыть`») и откройте файл «`Book.Mod`». Нажмите `F9` («`Собрать и запустить`»). Если всё работает как надо, Вы должны увидеть изображение книги.

Некоторые возможные неисправности:

1. «`Модуль Graph не найден`».

   Используется неправильная или неполная версия VOC. Модифицированная версия дополнена модулями `Graph` и `SDL2`, необходимыми для программирования графики. Убедитесь что файлы `Graph.sym`, `SDL2.sym`, `Graph.h`, `SDL2.h` и `SDL2.h0` находятся в правильных каталогах (sym-файлы в каталоге «`data/bin/voc/C/sym`», а остальные — в каталоге «`data/bin/voc/C/include`».
2. «`FLOOR — неопределённый идентификатор`».

   Используется неправильная версия VOC. В модифицированной версии VOC есть встроенный оператор `FLOOR(x)`, который, по существу, работает также, как `SHORT(ENTIER(x))`, переводя значение типа `REAL` или `LONGREAL` в значение типа `INTEGER` (а не `LONGINT`).

3. Ошибка «`.`».

   Команда «`voc`» недоступна или один из файлов «`data/bin/*.sh`» не помечен как исполнимый (в GNU/Linux).

3. Написание программы.

Запустите Free Oberon и наберите текст программного модуля. Модуль всегда начинается с ключевого слова `MODULE`, после которого следует его название. Название модуля должно быть написано слитно, начинаться с латинской буквы и содержать только латинские буквы и цифры. Затем пишется точка с запятой. Например, «`MODULE Prog1;`». Текст модуля заканчивается ключевым словом `END`, за которым (повторно) следует название модуля и точка:

```
MODULE Prog1;
  (*программа набирается между этими двумя строками*)
END Prog1.
```

4. Сохранение файла.

Чтобы сохранить файл, нажмите клавишу F2 или «File → Save As», после чего Вам будет предложено ввести имя файла. Чтобы модуль работал, файл должен называться также, как модуль, но с «.Mod» на конце (буква M должна быть заглавная). Например, «Prog1.Mod». Если название модуля и файла, в котором он находится, не совпадают, то откомпилированный модуль невозможно будет запустить из Free Oberon.

Последующие нажатия клавиши F2 будут сохранять модуль в файл с тем же именем. Чтобы сохранить модуль в файл с другим именем, нажмите «File → Save As...» или Shift+F2.

Сохранённые файлы являются обычными текстовыми файлами в кодировке UTF-8, находятся в подкаталоге «Programs» и, при желании, могут быть отредактированы другими текстовыми редакторами.

5. Запуск программы.

Чтобы скомпилировать и запустить программу, нажмите «Run → Compile & Run» или клавишу F9. Файл должен быть предварительно сохранён, а имя файла должно совпадать с названием модуля (см. выше «4. Сохранение файла»). Если файл не был сохранён, будет открыто диалоговое окно сохранения файла — сохраните файл, а затем нажмите F9 снова.

6. Открытие файла.

Нажмите F3 и введите имя файла, затем нажмите Ввод. Если файл существует, то он будет открыт, и в заголовке появившегося окна вы увидите его имя открытого файла. Если отредактировать файл и нажать «File → Save» или клавишу F2, то файл сохранится туда же. Скопировать файл можно сохранив его под другим именем, для этого нажмите «File → Save As...» или Shift+F2.

Пункт меню «File → Reload» полезна в том случае, если файл уже открыт и требуется перечитать его с диска (например, чтобы откатить неудачные изменения или файл на диске обновился).

Чтобы создать новый файл, нажмите «File → New» или Shift+F3.

7. Перемещение по тексту.

Перемещаться по тексту можно с помощью указателя мыши, но гораздо удобнее это делать с помощью клавиатуры.

Клавиши со стрелками позволяют перемещаться на один символ влево и вправо, а также на одну строку вверх и вниз. Чтобы быстро оказаться в начале строки, нажмите клавишу Home, чтобы в конце — нажмите клавишу End. С помощью клавиш

PageUp и PageDown можно переместиться вверх или вниз сразу на целый экран, что удобно при работе с большими файлами.

Клавиша Tab поставит один или два пробела, в зависимости от того, как далеко от левого края окна находится текстовый курсор. Это может быть удобно для расстановки отступов в тексте программы.

При нажатии клавиши Ввод, на текущей позиции вставляется новая строка, причём в неё автоматически добавляется столько же пробелов, сколько содержала предыдущая строка в начале (это называется «автоотступ»).

«Висячие» пробелы в конце каждой строки обозначаются точками.

8. Копирование строк.

Чтобы скопировать строку, её надо сначала выделить:

1. Переместите текстовый курсор в начало строки, которую требуется скопировать (для этого воспользуйтесь клавишами Вверх/Вниз, а также клавишей Home, которая быстро перемещает курсор в начало строки.
2. Зажмите клавишу Shift и, не отпуская её, один раз нажмите клавишу со стрелкой вниз. Курсор переместится на одну строку вниз, а копируемая строка окажется выделенной. (Таким же образом можно выделить и несколько строк.)
3. Нажмите Ctrl+C и строка скопируется в буфер обмена.
4. Нажмите Ctrl+V и на месте текстового курсора появится скопированная строка (она вставится из буфера обмена). Операцию Ctrl+V можно производить несколько раз.

Переместить строку можно аналогично, но вместо Ctrl+C нажмите Ctrl+X.

Если требуется просто удалить выделенный текст, нажмите «Edit → Clear» или клавишу Delete, или сочетание клавиш Ctrl+Del.

Чтобы выделить весь текст, нажмите Ctrl+A.

Все вышеупомянутые операции также доступны в меню «Edit».

9. Перемещение между открытыми окнами.

Если вы открыли несколько окон с программным текстом, можно легко перемещаться от одного окна к другому с помощью клавиши F6. Движение в обратном направлении осуществляется при помощи сочетания клавиш Shift+F6. Закрыть окно можно по Alt+F3, развернуть окно на весь экран и свернуть обратно — по F5. Все эти действия доступны в меню «Window», там же можно посмотреть и соответствующие горячие клавиши.

Перемещаться между окнами, менять их размер и закрывать их можно также и с помощью мышки — в нижнем правом углу окна есть специальный уголок, потянув за который, можно изменить размер окна, перемещать же окно можно за его заголовок.

10. Простейшая программа.

Включите Free Oberon и напишите следующую программу:

```
MODULE MyProg;
IMPORT In, Out;
VAR a, b, c: INTEGER;
BEGIN
  Out.String('Введите первое слагаемое: '); In.Int(a);
  Out.String('Введите второе слагаемое: '); In.Int(b);
  c := a + b;
  Out.String('Сумма равна '); Out.Int(c, 0); Out.Ln
END MyProg.
```

Сохраните файл как «`MyProg.Mod`» и нажмите клавишу `F9`. Если всё было введено правильно, программа запустится и запросит два числа, после чего отобразит их сумму и завершится.

Секция `IMPORT` описывает используемые модули. В данном примере используются модули `In` и `Out`. Модули перечисляются через запятую и каждый из них может быть использован далее в тексте программы.

В секции `VAR` перечисляются переменные и их типы. Переменная — это именованный блок памяти, в котором хранится некоторое значение (каждая переменная имеет название). Тип переменной описывает множество значений, которые может принимать (т. е. иметь) переменная. В данном примере объявлены три переменные: `a`, `b` и `c`, и все три имеют тип `INTEGER`, т. е. «целое число».

После ключевого слова `BEGIN` следуют команды, которые программа будет выполнять. Между каждыми двумя смежными командами стоит точка с запятой. В конце последней команды точка с запятой допускается, но не является обязательной, и поэтому мы её ставить не будем (см. команду `Out.Ln` в тексте программы).

После некоторых команд в скобках указываются передаваемые им *параметры*. Например, после `Out.String` в скобках указан некоторый текст (в кавычках). Этот текст *передаётся* команде `Out.String`, а она выводит его на экран.

Команда `In.Int` приостанавливает выполнение программы, пока человек не введёт число и не нажмёт клавишу Ввод, после чего введённое значение записывается в указанную в скобках переменную.

Команда «`c := a + b`» — так называемый *оператор присваивания*. Он вычисляет значение справа от «`:=`», а получившееся значение записывает в переменную, указанную слева. Команду «`c := a + b`» можно воспринять следующим образом: «Вычислить сумму *a* + *b* и записать её в переменную c».

`Out.Int(c, 0)` выводит число `c` на экран. Второй параметр `0` указывает минимальное число знаков, которое выводимое число должно занимать на экране. Например, `Out.Int(14, 5)` выведет на экран три пробела и число 14, в результате чего будет выведено 5 знаков.

`Out.Ln` выполняет две функции. Она переводит строку (так, что следующий текст будет выведен на следующей строке) и обеспечивает то, что текст, выведенный ранее становится видимым на экране. Это значит, что если не выполнить команду `Out.Ln` в конце программы, то, возможно, на экране будет отображён не весь выведенный ранее текст.

11. Основные типы данных.

Каждая переменная должна иметь определённый тип. Тип задаёт множество возможных значений. Существуют следующие основные типы:

`INTEGER` — целое число в промежутке от −2 147 483 648 до 2 147 483 647.
`REAL` — вещественное (дробное) число в промежутке от 10^{-38} до 10^{38}.
`CHAR` — один символ (литера), т. е. одна буква, цифра, пробел и т. д.
`BOOLEAN` — логический (булевый) тип, принимает значения `TRUE` или `FALSE`.
`SET` — множество, может содержать целые числа от 0 до 31.

12. Консольный ввод и вывод. Модули In, Out.

Консольный ввод, как правило, ведётся с клавиатуры, а консольный вывод отражается на экране. В языке Оберон консольный ввод осуществляется при помощи модуля `In`, а консольный вывод — через модуль `Out`.

Модуль `In` содержит следующие *процедуры*:

```
Int(VAR i: INTEGER)                 — ввод целого числа
Real(VAR x: REAL)             — ввод вещественного числа
Char(VAR ch: CHAR)                  — ввод одного символа (литеры)
Line(VAR str: ARRAY OF CHAR)  — ввод текстовой строки
```

Модуль `Out` содержит следующие процедуры:

`Int(i, n: INTEGER)`	Выводит целое число *i* так, чтобы оно занимало не менее *n* знаков, по необходимости добавляя слева пробелы.
`Hex(i, n: INTEGER)`	То же, что и `Int`, но число выводится в шестнадцатеричном виде.
`Real(x: REAL; n: INTEGER)`	Выводит вещественное число (шириной в *n* знаков) в научном виде.
`RealFix(x: REAL; n, k: INTEGER)`	Выводит вещественное число в виде десятичной дроби (шириной в *n* знаков) с *k* знаками после запятой.
`Char(VAR ch: CHAR)`	Выводит один символ (литеру)
`String(str: ARRAY OF CHAR)`	Выводит текстовую строку

`Ln`	Осуществляет переход на следующую строку, перед чем сбрасывает буфер вывода (т. е. делает `Flush`).
`Flush`	Сбрасывает буфер вывода. В результате этой команды весь ранее выведенный текст отображается на экране.

Примеры:
```
Out.String('Hello '); Out.Int(123, 0); Out.Int(a, 4); Out.Ln;
Out.String('x = '); Out.RealFix(x, 0, 2); Out.Flush
```

13. Модуль Math.

Модуль `Math` содержит некоторые математические функции и обычно импортируется под псевдонимом «`M`»:
```
IMPORT M := Math;
```

Некоторые процедуры модуля `Math`:
`round(x: REAL): LONGINT` — число *x*, округлённое до ближайшего целого,
`sqrt(x: REAL): REAL` — корень квадратный числа *x*,
`exp(x: REAL): REAL` — число *e*, возведённое в степень *x*,
`ln(x: REAL): REAL` — натуральный логарифм числа *x*,
`sin(x: REAL): REAL` — синус угла *x*, выраженного в радианах,
`cos(x: REAL): REAL` — косинус угла *x*, выраженного в радианах,
`tan(x: REAL): REAL` — тангенс угла *x*, выраженного в радианах,
`arcsin(x: REAL): REAL` — арксинус числа *x*, выраженный в радианах,
`arccos(x: REAL): REAL` — арккосинус числа *x*, выраженный в радианах,
`arctan(x: REAL): REAL` — арктангенс числа *x*, выраженный в радианах,
`power(base, exp: REAL): REAL` — число *base*, возведённое в степень *exp*,
`ipower(x: REAL; base: INTEGER): REAL` — число *x* в (целой) степени *base*,
`log(x, base: REAL): REAL` — логарифм числа *x* по основанию *base,*
`sincos(x: REAL; VAR Sin, Cos: REAL)` — синус и косинус угла *x* (в рад.),
`arctan2(xn, xd: REAL): REAL` — арктангенс частного *xn/xd* (рад.),
а также гиперболические функции: `sinh(x), cosh(x), tanh(x), arcsinh(x), arccosh(x), arctanh(x).`

14. Модуль Graph.

Модуль `Graph` позволяет программировать графику и звук, более углублённо взаимодействовать с клавиатурой, а также генерировать псевдослучайные числовые последовательности. Модуль Graph обычно импортируется под псевдонимом «`G`»:
```
IMPORT G := Graph;
```

Простейший графический пример:

```
MODULE GrTest;
IMPORT G := Graph;
VAR screen: G.Bitmap;
BEGIN
```

```
  screen := G.Init();
  G.Line(screen, 20, 30, 150, 100, G.MakeCol(255, 0, 0));
  G.Flip;
  G.Pause;
  G.Close
END GrTest.
```

Данная программа откроет (чёрное) графическое окно, нарисует на нём красный отрезок (`G.Line`) из точки (20; 30) в точку (150; 100), подождёт нажатия человеком любой клавиши (`G.Pause`), после чего завершится.

`G.Init` инициализирует (начинает) графику, т. е. открывает графическое окно и *возвращает* указатель на него, этот указатель сохраняется в переменной `screen` типа `G.Bitmap`, а затем используется для рисования.

`G.Flip` отображает нарисованное изображение на экране. Пока не вызвана процедура `G.Flip`, картинки не видно.

`G.Pause` приостанавливает выполнение программы и ждёт, пока человек нажмёт на клавиатуре какую-либо клавишу.

`G.Close` закрывает графическое окно.

14.1. Рисование отрезков.

Процедура `G.Line` чертит отрезок. Она принимает шесть параметров. Первый параметр указывает, где рисовать отрезок (на экране), четыре последующих параметра — это координаты начала и конца отрезка: $(x_1; y_1)$, $(x_2; y_2)$, они передаются как четыре целых числа через запятую. Координаты отсчитываются от левого верхнего угла экрана — точки (0; 0) и увеличиваются вправо по оси *OX* и вниз по оси *OY*. Последний параметр — это цвет отрезка (см. «14.2. Задание цвета»).

Если отрезок горизонтальный, то его можно нарисовать, используя процедуру `G.HLine(screen, x1, y, x2, color)`, в которую значение *y* передаётся только один раз. Аналогично, для вертикального отрезка можно использовать процедуру `G.VLine(screen, x, y1, y2, color)`.

14.2. Задание цвета.

Цвет задаётся с помощью процедуры `G.MakeCol(r, g, b)`, которая принимает три целых числа, соответствующих красной, зелёной и синей составляющей. Каждое число лежит в промежутке от 0 до 255. Ниже показано, как можно получить некоторые цвета:

```
G.MakeCol(  0,   0,   0)          — чёрный
G.MakeCol(255, 255, 255)          — белый
G.MakeCol( 80,  80,  80)          — тёмно-серый
G.MakeCol(  0,   0, 255)          — синий
G.MakeCol(  0, 255, 255)          — циан (ядовито сине-зелёный)
G.MakeCol(120,  60,   0)          — коричневый
G.MakeCol(255, 229, 180)          — персиковый
```

Сам получившийся цвет тоже имеет тип `INTEGER`, поэтому его можно записать в переменную, а затем использовать вместо `G.MakeCol(...)`:

```
VAR orange: INTEGER;
BEGIN
  orange := G.MakeCol(255, 128, 0);
  G.Line(screen, 100, 100, 200, 100, orange)
```

Чтобы разложить цвет на его составляющие, есть процедура `ColorToRGB`:

```
VAR color, r, g, b: INTEGER;
BEGIN
  color := G.MakeColor(0, 128, 255);
  G.ColorToRGB(color, r, g, b)
```

14.3. Получение размера экрана.

После того, как графика была инициализирована `screen := G.Init()`, становятся доступны два значения: `screen.w` и `screen.h`, где первое выражает ширину экрана в пикселях, а второе — высоту. Например, перечеркнуть экран двумя линиями можно следующим образом:

```
G.Line(screen, 0, 0, screen.w - 1, screen.h - 1, G.MakeCol(255, 0, 0));
G.Line(screen, screen.w - 1, 0, 0, screen.h - 1, G.MakeCol(0, 255, 0))
```

Следует отметить, что самый правый пиксель, который виден на экране имеет абсциссу `screen.w - 1`, потому что абсцисса самого левого — 0. Подобное справедливо и для `screen.h`.

14.4. Манипулирование цветом отдельных пикселей.

Можно изменить цвет одной конкретной точки:

```
G.PutPixel(screen, 100, 60, G.MakeCol(0, 0, 255))
```

Узнать цвет точки можно с помощью процедуры `GetPixel`.

```
VAR color: INTEGER;
BEGIN
  color := G.GetPixel(screen, 100, 60)
```

Если же требуется очень быстро менять цвет отдельных пикселей, то лучше воспользоваться процедурой `PutPixelFast`. Для этого необходимо сначала вызвать процедуру `LockBitmap`, а в конце вызвать `UnlockBitmap`:

```
LockBitmap(screen);
PutPixelFast(screen, x, y, color);
UnlockBitmap(screen)
```

14.5. Очистка экрана.

Чтобы закрасить весь экран чёрным цветом, можно вызвать процедуру `G.ClearScreen`. Чтобы очистить экран каким-нибудь другим цветом, можно воспользоваться процедурой `G.ClearScreenToColor`:

```
G.ClearScreenToColor(G.MakeCol(0, 0, 80))
```

14.6. Рисование некоторых фигур.

С помощью процедуры `G.Rect` можно нарисовать прямоугольную рамку:
```
G.Rect(screen, 50, 100, 200, 150, G.MakeCol(255, 0, 0))
```

Вторым и третьим параметром передаются координаты левого верхнего угла, а четвёртым и пятым — координаты правого нижнего угла. В примере будет нарисован прямоугольник шириной в 151 и высотой в 51 пиксель.

Процедура `G.RectFill` рисует закрашенный прямоугольник.

14.7. Настройка графического окна.

Перед вызовом `G.Init`, можно задать некоторые параметры графического окна с помощью вызова процедуры `G.Settings(w, h, flags)`. Первые два параметра — это желаемая ширина и высота экрана в пикселях, `flags` — это множество, в которое можно включить следующие константы:

`G.fullscreen` — включить графику в полноэкранном режиме
`G.spread` — если пропорции позволяют, увеличить размер экрана
`G.sharpPixels` — обеспечить кратный размер виртуального пикселя (1, 2, 3, 4...)
`G.software` — отключить аппаратную прорисовку
`G.initMouse` — инициализировать автоматическое управление мышью

Пример:
```
G.Settings(320, 200, {G.fullscreen, G.spread, G.sharpPixels});
screen := G.Init()
```

Размер окна по умолчанию — 640 на 400 пикселей, но фактически размер может оказаться бóльшим, т. к. по умолчанию включены настройки `G.fullscreen` и `G.spread` (а также `G.sharpPixels`). Если в качестве ширины или высоты задать нуль и указать `G.fullscreen`, то графическое окно просто займёт весь весь экран.

Когда окно уже инициализировано, можно перейти в оконный режим с помощью `G.SwitchToWindowed`, а в полноэкранный режим — с помощью `G.SwitchToFullscreen`. Процедура же `G.ToggleFullscreen` переключает графическое окно туда и обратно.

14.8. Анимация.

Анимация подразумевает быструю смену кадров, поэтому необходим цикл, в котором программа будет прорисовывать очередной кадр, а затем отображать его на экране с помощью `G.Flip` и, возможно, делать небольшую задержку с помощью процедуры `G.Delay`. Цикл можно прервать по нажатию клавиши — `G.KeyPressed()` или по какому-либо иному признаку. Пример:

```
MODULE FlyingDot;
IMPORT G := Graph;
VAR s: G.Bitmap;
  x, y, vy: INTEGER;
BEGIN
  s := G.Init();
```

```
  x := 0; y := 10; vy := 0;
  REPEAT
    G.PutPixel(s, x, y, G.MakeCol(255, 255, 255));
    INC(x, 2); INC(y, vy); INC(vy);
    IF vy > 15 THEN vy := -13 END;
    G.Flip;
    G.Delay(20)
  UNTIL G.KeyPressed();
  G.Close
END FlyingDot.
```

Процедура `G.Delay` принимает целое число — количество миллисекунд, которое необходимо выждать (в одной секунде 1000 миллисекунд). В конце вызова процедуры `G.KeyPressed()` ставятся скобки, она возвращает значение типа `BOOLEAN`. Если была нажата клавиша, то она возвратит `TRUE` и цикл `REPEAT` завершится.

14.9. Генератор псевдослучайных последовательностей чисел.

Для программирования графики часто оказывается необходим генератор псевдослучайных чисел. Он запускается с помощью процедуры `G.Randomize`, но `G.Init` также запускает генератор, поэтому вызывать `G.Randomize` нет необходимости.

Процедура `G.Random(n)` возвращает случайное целое число в промежутке от 0 до (n – 1) (включительно). Если Вам необходимо число от 1 до n, воспользуйтесь выражением `G.Random(n) + 1`.

Процедура `G.Uniform()` возвращает случайное число в промежутке [0; 1). Если его затем домножить на n, то получится промежуток [0; n).

Пример:

```
VAR a: INTEGER;
  x: REAL;
BEGIN
  a := G.Random(10);      (* целое число в промежутке [0; 9]   *)
  x := G.Uniform() * 9;   (* дробное число в промежутке [0; 9) *)
```

Псевдослучайная последовательность основывается на *случайном зерне*, которое устанавливается в начале программы при вызове `G.Randomize`, и меняется каждый раз при генерации следующего случайного числа. Это зерно также можно задать и явно, с помощью процедуры `G.PutSeed(n)`. Такой приём может быть использован для повторной генерации одних и тех же псевдослучайных последовательностей. Случайное зерно имеет тип `INTEGER` и доступно для чтения посредством `G.randomSeed`.

14.10. Работа с изображениями.

Процедура `G.LoadBitmap(filename)` позволяет загрузить изображение из файла в формате `BMP`, `PNG` или `JPG`. Процедура возвращает указатель типа `G.Bitmap`. Затем, это изображение можно отрисовать на экране или на другом изображении с помощью процедур `G.Blit`, `G.BlitWhole` и `G.StretchBlit`. Пример:

```
MODULE BlitBmp;
IMPORT G := Graph;
VAR s, b: G.Bitmap;
BEGIN
  s := G.Init();
  b := G.LoadBitmap('data/examples/rocket.png');
  G.BlitWhole(b, s, 100, 60);
  G.Flip; G.Pause; G.Close
END BlitBmp.
```

Процедура `BlitWhole` отрисовывает изображение в заданных координатах, `Blit` может отрисовать часть изображения, а `StretchBlit` может растягивать и сжимать изображение при отрисовке.

```
BlitWhole(src, dest: Bitmap; x, y: INTEGER)
Blit(src, dest: Bitmap; sx, sy, sw, sh, dx, dy: INTEGER)
StretchBlit(src, dest: Bitmap; sx, sy, sw, sh, dx, dy, dw, dh: INTEGER)
```

Параметр `src` (от англ. source) задаёт *откуда* брать изображение, а `dest` (от англ. destination) — *куда*. Параметры `sx`, `sy`, `sw` и `sh` относятся к `src`, а `dx`, `dy`, `dw` и `dh` — к `dest`.

Пример:

```
MODULE StretchBlitBmp;
IMPORT G := Graph;
VAR s, b: G.Bitmap;
BEGIN
  s := G.Init();
  b := G.LoadBitmap('data/examples/rocket.png');
  G.StretchBlit(b, s, 0, 0, b.w, b.h, 0, 0, s.w, s.h);
  G.Flip; G.Pause; G.Close
END StretchBlitBmp.
```

Программа непропорционально растянет изображение ракеты на весь экран.

Приложение А. Пересборка Free Oberon под ОС Windows.

Несмотря на то, что Free Oberon поставляется в виде EXE-файла со всеми необходимыми дополнениями, в некоторых случаях имеет смысл скомпилировать его заново — например, после внесения изменений в код Free Oberon или в целях самообразования. Для этого Вам потребуется установить некоторое программное обеспечение.

Прежде всего, это компилятор Си «MinGW-w64» и Юникс-подобное окружение «MSYS2», далее транслятор Оберона в Си «Vishap Oberon Compiler» и, наконец, графическая библиотека SDL2 с дополнениями.

1. Перейдите по ссылке `sourceforge.net/projects/mingw-w64` и скачайте MinGW-w64 — компилятор GCC для ОС Windows (несмотря на «64» в названии, данная программа работает в 32-разрядном режиме).
2. Установите MinGW-w64 в каталог «`C:\mingw-w64`». Данный путь потребуется нам позже.

Примечание. Если во время установки Вы сменили диск `C:` на другой и MinGW-w64 не устанавливается, попробуйте использовать диск `C:`.

3. Перейдите по ссылке `msys2.org` и скачайте версию MSYS2 для архитектуры процессора i686 (имя скачиваемого файла скорее всего будет иметь вид «`msys2-i686-*.exe`»). Если Вы используете Windows XP, то новейшая версия MSYS2 работать не будет, и Вам следует скачать более старую версию, например, «`msys2-i686-20150916.exe`».
   Её можно скачать по адресу:
   `sourceforge.net/projects/msys2/files/Base/i686`.

   Если и это не помогает, скачайте архив в формате «`tar.xz`» и разархивируйте его с помощью программы 7-Zip (7zip.org).

4. Установите MSYS2 в каталог «`C:\msys32`» и запустите терминал MSYS. Его можно запустить, используя файл «`C:\msys32\msys32_shell.bat`».

5. В терминале MSYS запустите команду:
   ```
   pacman -Sy pacman
   ```

   Эта команда обновляет базу данных пакетов внутри MSYS, а также обновляет сам pacman. (Когда программа спросит, приступить ли к установке, ответьте утвердительно — «`y`».)

6. Закройте терминал MSYS, а затем откройте его снова (см. пункт 4).

7. Обновите остальную систему, запустив в терминале MSYS команду:
   ```
   pacman -Su
   ```

8. Снова закройте терминал MSYS и откройте его с правами администратора. (Щёлкните правой кнопкой мыши по `msys32_shell.bat` или по иконке в меню приложений и выберите «Run as administrator».)

9. Установите программы `make` и `diffutils`:
   ```
   pacman -S make diffutils
   ```

10. Установите переменные окружения следующими командами:
    ```
    export PATH=$PATH:/c/mingw-w64/mingw32/bin
    export CC=gcc
    ```

    Это необходимо, чтобы в процессе компиляции был использован компилятор MinGW. Если в пункте 2 выше Вы указали другой путь, то и здесь его необходимо скорректировать.

    Эти две строки можно дописать в файл «`~/.bashrc`», тогда они будут автоматически срабатывать каждый раз при запуске терминала MSYS2.

11. Перейдите по ссылке libsdl.org/download-2.0.php, скачайте архив «`SDL2-devel-2.*.*-mingw.tar.gz`», разархивируйте и зайдите в него. Скопируйте содержимое каталога «`i686-w64-mingw32`» в «`C:\mingw-w64\mingw32`».

Скопируйте файл «bin\SDL2.dll» из распакованного архива в каталог «C:\FreeOberon».

12. Перейдите по ссылке libsdl.org/projects/SDL_image и скачайте архив «SDL2_image-devel-2.*.*-mingw.tar.gz», разархивируйте и зайдите в него. Скопируйте содержимое каталога «i686-w64-mingw32» в «C:\mingw-w64\mingw32». По желанию можно аналогично произвести установку других дополнений к библиотеке SDL2.

    Также зайдите в подкаталог «i686-w64-mingw32\bin» распакованного архива SDL2 image и скопируйте следующие DLL-файлы: libjpeg-9.dll, libpng16-16.dll, SDL2_image.dll и zlib1.dll в каталог «C:\FreeOberon».

    Примечание. Если Вы хотите запускать *графические* приложения из «C:\FreeOberon\bin» вручную, скопируйте DLL файлы и в этот каталог.

    Пересборка завершена.

13. Перейдите в каталог с исходным кодом и запустите компиляцию:
    ```
    cd /c/FreeOberon/src
    make -f Makefile_win32
    ```

    В результате должен быть собран компилятор VOC и библиотеки для него, а также среда Free Oberon.

    Чтобы удалить не нужные для работы файлы, запустите:
    ```
    make -f Makefile_win32 clean
    ```

14. Чтобы пересобрать всё заново, запустите следующие команды в MSYS (но не забудьте установить переменные окружения, см. шаг 10):
    ```
    cd /c/FreeOberon/src
    make -f Makefile_win32
    ```