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

с исправлениями от 8 ноября 2023 г.
для версии Free Oberon 1.1.0

ИЗДАТЕЛЬСТВО
ТЕХНИКО-ТЕОРЕТИЧЕСКОЙ ЛИТЕРАТУРЫ
РИЖСКОГО ОКБ КОМПИЛЯТОРОСТРОЕНИЯ

Р И Г А 2 0 2 3

Оглавление

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 исходного кода, которые необходимо скомпилировать, предварительно снабдив операционную систему необходимыми библиотеками.

1.1. Установка под ОС GNU/Linux.

1. Скачайте исходные коды Free Oberon. Есть два варианта:

   a) Склонируйте репозиторий Free Oberon с GitHub. Эта версия наиболее свежая. Для этого наберите в терминале:

   ```
   git clone --depth 1 --recurse-submodules --shallow-submodules \
     https://github.com/kekcleader/FreeOberon.git
   ```

   > Примечание: обратная дробь (\) экранирует перенос строки. Можно вводить команду без «\», если команда вводится в одну строчку.

   b) Скачайте Free Oberon в виде архива «tar.gz» с раздела «Скачать» сайта free.oberon.org. Распакуйте архив в свой домашний каталог или в другое место на диске.

2. Установите следующие пакеты программного обеспечения:

   ```
   git         gcc         libc-dev                liballegro5-dev
   liballegro-image5-dev   liballegro-audio5-dev   liballegro-acodec5-dev
   ```

   Имена пакетов даны в соответствии с их наименованием в ОС Debian GNU/Linux. Они подойдут для пользователей Ubuntu, Linux Mint, Raspbian и др. Для установки указанных пакетов в терминале выполните команду:

   ```
   apt-get install -y git gcc libc-dev liballegro5-dev \
     liballegro-image5-dev liballegro-audio5-dev liballegro-acodec5-dev
   ```

   (Эту команду необходимо выполнить с правами суперпользователя, т. е. предварительно необходимо запустить «`su`» и ввести пароль, или приписать в начале команды слово «`sudo`».)

   На ОС Fedora, Red Hat, CentOS и других, команда и имена пакетов будут отличаться (в некоторых случаях необходимо будет установить один из двух пакетов: `glibc-static` или `glibc-devel-static`):

   ```
   sudo dnf install -y git gcc glibc-devel allegro5-devel \
     allegro5-addon-image allegro5-addon-audio allegro5-addon-acodec \
     allegro5-devel allegro5-addon-image-devel allegro5-addon-audio-devel \
     allegro5-addon-acodec-devel
   ```

   Для других операционных систем (Arch Linux, openSUSE) обратитесь к выводу команды «`./install.sh`» в каталоге распакованного архива Free Oberon.

3. Запустите:

   ```
   ./install.sh
   ```

4. Готово. Если Free Oberon был успешно установлен, Вы можете запустить его с помощью «`./FreeOberon`». Установщик предложит Вам по желанию дописать в файл «`~/.bashrc`» строчку, которая позволит использовать команду «`fob`» (консольную версию компилятора Free Oberon) — примерно такую строку:

   ```
   export PATH=/home/user/FreeOberon:$PATH
   ```

   Можно добавить ещё и следующую строку, которая позволит запускать Free Oberon, находясь в любом каталоге, при помощи команды «`fo`»:

   ```
   alias fo='cd ~/FreeOberon;./FreeOberon'
   ```

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`» в подкаталоге «`Examples`». Нажмите `F9` («`Собрать и запустить`»). Если всё работает как надо, Вы должны увидеть изображение книги.

3. Выбор языка.

Среда Free Oberon откроется на языке вашего окружения операционной системы: английском или русском. Есть возможность указать язык. Соответствующий аргумент командной строки «`--lang`». Например:

```
./FreeOberon --lang en     (в терминале Линукс)
FreeOberon --lang en       (в командной строке Windows)
```

А ещё можно добавить свой язык. См. Текстовые файлы в подкаталоге «`Data/Texts`».

4. 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. Имя файла: Prog1.Mod *)
END Prog1.
```

5. Сохранение файла.

Чтобы сохранить файл, нажмите клавишу F2 или «Файл → Сохранить как...», 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 «Файл → Сохранить как...» или Шифт+F2.

Сохраняемые файлы — простые текстовые файлы в кодировке UTF-8, по умолчанию они помещаются в подкаталог «Programs» и, при желании, могут быть отредактированы другими текстовыми редакторами.

6. Запуск программы.

Чтобы скомпилировать и запустить программу, нажмите «Компиляция → Смастерить и запустить» или клавишу F9. Файл будет сохранён, его имя должно соответствовать названию модуля (см. § 4 «Сохранение файла»). Если файл не был сохранён, будет открыто диалоговое окно сохранения — сохраните файл, а затем снова нажмите F9.

7. Открытие файла.

Нажмите F3 и выберите файл, затем нажмите Ввод. Файл будет открыт, и в заголовке появившегося окна вы увидите его имя. Если отредактировать файл и нажать «Файл → Сохранить» или клавишу F2, то файл сохранится туда же, откуда был прочитан. Скопировать же файл можно, сохранив его под другим именем, для этого нажмите «Файл → Сохранить как...» или Шифт+F2.

Пункт меню «Файл → Reload» полезен в том случае, если файл был изменён и его требуется повторно считать с диска (например, чтобы откатить неудачные изменения или в случае, если файл на диске был изменён другой программы).

Чтобы создать новый файл, нажмите «Файл → Новый» или Шифт+F3.

8. 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.

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 Домой (Home), которая быстро перемещает курсор в начало строки.
2. Зажмите клавишу Шифт и, не отпуская её, один раз нажмите клавишу со стрелкой вниз. Курсор переместится ниже на одну строку, а строка окажется выделенной. (Таким же образом можно выделить и несколько строк.)
3. Нажмите Ктрл+C и строка скопируется в буфер обмена.
4. Нажмите Ктрл+V и на месте текстового курсора появится скопированная строка (она вставится из буфера обмена). Операцию Ктрл+V можно производить несколько раз.

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

Если требуется просто удалить выделенный текст, нажмите «Правка → Удалить» или клавишу Удалить (Delete).

Чтобы выделить весь текст, нажмите Ктрл+A.

Все вышеупомянутые операции также доступны в меню «Правка».

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 Шифт+F6. Закрыть окно можно по Альт+F3, развернуть окно на весь экран и свернуть

обратно — по F5. Все эти действия доступны в меню «Окно», там же можно посмотреть и соответствующие горячие клавиши.

Перемещаться между окнами, менять их размер и закрывать их можно также и с помощью мыши — 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 запросит два числа, 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 имя). Тип переменной описывает множество значений, которые может принимать переменная. Здесь объявлено три переменных: a, b и c, и все три имеют тип INTEGER — целое число.

После ключевого слова BEGIN следуют команды, которые программа будет выполнять. Между любыми двумя смежными командами стоит точка с запятой. В конце последней команды точка с запятой допустима, но не обязательна, 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. Базовые типы данных.

Каждая переменная должна иметь определённый тип. Тип данных определяет набор значений, которые могут принимать переменные этого типа, и операторы, которые могут быть применены к ним. Существуют следующие базовые 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.

А также составной тип:
`POINTER TO` … — указатель, хранящий адрес в памяти какого-либо другого значения.

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. Для краткости кода его можно импортировать под псевдонимом «`M`»:

```
IMPORT M := Math;
```

Некоторые процедуры модуля `Math`:
`round(x: REAL): INTEGER` — 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 Random.

Генератор псевдослучайных числовых последовательностей реализован в модуле `Random`. Он инициализируется с помощью процедуры `Randomize`, но она вызывается автоматически при импорте модуля, поэтому вызывать `Random.Randomize` вручную 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`. Случайное целое число в промежутке [a; b] получается в общем виде выражением `G.Random(b - a + 1) + a`.

Процедура `G.Uniform()` возвращает случайное вещественное 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 `Random.seed`.

15. Модуль Graph.

Модуль `Graph` позволяет программировать графику, взаимодействовать с клавиатурой и мышью. Модуль Graph обычно импортируется под псевдонимом «`G`»:

```
IMPORT G := Graph;
```

Простейший пример графики:

```
MODULE GrTest;
IMPORT G := Graph;
VAR c: G.Color;
BEGIN
  G.Init;
  G.MakeCol(c, 255, 0, 0);
  G.Line(20, 30, 150, 100, c);
  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 (запускает) графику, т. е. открывает графическое окно.

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

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

`G.Close` закрывает графическое окно.

15.1. Рисование отрезков.

Процедура `G.Line` чертит отрезок. Она принимает пять параметров. Первые четыре параметра — это координаты начала и конца отрезка в следующем порядке:

$(x_1; y_1)$, $(x_2; y_2)$, 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. § 15.2 «Задание цвета»).

Если отрезок горизонтальный, то его можно нарисовать процедурой `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)`.

15.2. Задание цвета.

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

```
G.MakeCol(c,   0,   0,   0)        — чёрный
G.MakeCol(c, 255, 255, 255)        — белый
G.MakeCol(c,  80,  80,  80)        — тёмно-серый
G.MakeCol(c,   0,   0, 255)        — синий
G.MakeCol(c,   0, 255, 255)        — бирюзовый
G.MakeCol(c, 120,  60,   0)        — коричневый
G.MakeCol(c, 255, 229, 180)        — персиковый
```

Здесь `c` — переменная типа `G.Color`, в которую и записывается получившийся цвет.

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

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

15.3. Получение размера экрана.

После того, как графика была инициализирована процедурой `G.Init`, можно вызвать процедуру `G.GetScreenSize(W, H)`, где `W` и `H` — переменные типа `INTEGER`. В первую переменную будет помещена ширина экрана (в пикселях), во вторую — высота.

Перечеркнуть экран двумя линиями можно следующим образом:

```
MODULE SizeTest;
IMPORT G := Graph;
VAR c: G.Color;
  W, H: INTEGER;
BEGIN
  G.Init;
  G.GetScreenSize(W, H);
  G.MakeCol(c, 0, 255, 0);
  G.Line(0, 0, W - 1, H - 1, c);
```

```
  G.Line(0, H - 1, W - 1, 0, c);
  G.Flip; G.Pause; G.Close
END SizeTest.
```

Следует отметить, что самый правый пиксель, который виден на экране имеет абсциссу `W - 1`, потому что абсцисса самого левого — 0. Нижний пиксель на `H - 1`.

15.4. Манипулирование цветом отдельных пикселей.

Можно изменить цвет одной конкретной точки:

```
G.PutPixel(100, 60, color)         (* X, Y и цвет *)
```

Узнать цвет точки можно с помощью процедуры `GetPixel`.

```
VAR c: G.Color;   (* цвет *)
  bmp: G.Bitmap;  (* растр - массив пикселей *)
BEGIN
  G.Init;
  bmp := G.GetTarget();         (* Получаем растр экрана, записываем в bmp *)
  G.GetPixel(bmp, 250, 10, c)   (* Записываем цвет пикселя (250; 10) в c   *)
```

Если требуется многократно использовать процедуры `PutPixel` и `GetPixel`, можно существенно ускорить их работу, если предварительно заблокировать растр, а в конце разблокировать его. Для этого необходимо сначала вызвать процедуру `LockBitmap`, а в конце вызвать `UnlockBitmap`:

```
G.LockBitmap(bmp);
FOR x := 0 TO 100 DO
  G.GetPixel(bmp, x, y, c);
  (*...*)
END;
G.UnlockBitmap(bmp)
```

15.5. Очистка экрана.

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

```
G.ClearToColor(c)
```

15.6. Рисование некоторых фигур.

С помощью процедуры `G.Rect` можно нарисовать прямоугольную рамку:

```
G.Rect(50, 100, 200, 150, c)
```

Первым и вторым параметром передаются координаты некоторого верхнего угла прямоугольника, а третьим и четвёртым — координаты противоположного угла. В примере будет нарисован прямоугольник шириной в 151 и высотой в 51 пиксель.

Процедура `G.RectFill` рисует закрашенный прямоугольник.

15.7. 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.window`	— открыть графику в окне (а не на весь экран)
`G.center`	— открыть окно в центре (работает только в оконном режиме)
`G.resizable`	— сделать окно изменяемым в размере (оконный режим)
`G.exact`	— не увеличивать размеры экрана автоматически
`G.smooth`	— не обеспечивать специально чёткость виртуального пикселя
`G.software`	— отключить аппаратную прорисовку
`G.noMouse`	— не инициализировать мышь
`G.maximized`	— открыть окно максимально развёрнутым (оконный режим)
`G.minimized`	— открыть окно свёрнутым (оконный режим)
`G.frameless`	— открыть окно без рамки ОС (оконный режим)
`G.nobuffer`	— выключить двойную буферизацию

Пример:

```
G.Settings(320, 200, {G.exact, G.smooth, G.noMouse});
G.Init;
```

Размер окна по умолчанию — 640 на 400 пикселей, но фактически размер может оказаться бóльшим, т. к. по умолчанию он подстраивается в сторону увеличения для занятия всего экрана, одновременно обеспечивая, чтобы размер виртуальных пикселей был чилом целым (1, 2, 3 и т. д.). Если в качестве ширины или высоты задать нуль, графическое окно просто займёт весь экран, а виртуальные пиксели будут совпадать с реальными пикселями экрана.

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

15.8. 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 c: G.Color;
  x, y, vy: INTEGER;
BEGIN
  G.Init;
  G.MakeCol(c, 255, 255, 255);
```

```
  x := 0; y := 10; vy := 0;
  REPEAT
    G.PutPixel(x, y, c);
    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.

15.9. 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.Draw`, `G.DrawPart`, `G.DrawTintedPart`, `G.DrawFlip`, `G.DrawPartFlip`, `G.DrawRotated`, `G.DrawScaledRotated` и `G.DrawEx`.

Пример:

```
MODULE Rocket1;
IMPORT G := Graph, Out;
VAR b: G.Bitmap;
BEGIN
  G.Init();
  b := G.LoadBitmap('Data/rocket.png');
  IF b = NIL THEN Out.String('Could not load rocket.png'); Out.Ln
  ELSE
    G.Draw(b, 100, 60);
    G.Flip; G.Pause; G.Close
  END
END Rocket1.
```

Процедура `Draw` отрисовывает изображение в заданных координатах, `DrawPart` может отрисовать часть изображения, а `DrawEx` может растягивать и сжимать изображение при отрисовке.

```
Draw(bmp: Bitmap; x, y: INTEGER);
DrawPart(bmp: Bitmap; sx, sy, sw, sh, dx, dy: INTEGER);
DrawEx(bmp: Bitmap; sx, sy, sw, sh, dx, dy, dw, dh: INTEGER; flip: SET);
```

Значения *sx, sy, sw, sh* соответствуют к исходному изображению (что́ будет нарисовано), а *dx, dy, dw, dh* соответствуют целевому изображению (где будет нарисовано). Множество `flip` может быть пустым (`{}`) или может содержать одно или оба значений из множества `{flipHorz, flipVert}` для горизонтальнго и вертикального переворачивания исходного изображения соответственно.

```
G.DrawEx(b, 0, 0, b.w, b.h, 50, 50, b.w, b.h, {G.flipVert})
```

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 Allegro5 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. Скачайте набор разработчика Allegro5. Скопируйте содержимое подкаталога «`i686-w64-mingw32`» архива в «`C:\mingw-w64\mingw32`».

    Скопируйте файл «`allegro-5.2.dll`» из архива в каталог «`C:\FreeOberon`».

12. Сделайте то же с «`allegro-primitives-5.2.dll`» и «`allegro-image-5.2.dll`».

13. Перейдите в каталог с исходным кодом и запустите компиляцию:

    ```
    cd C:\FreeOberon\src
    make.bat
    ```

    В результате должен быть собран Free Oberon и библиотеки для него, а также консольный компилятор Free Oberon — *foc.exe*.