МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РОССИЙСКОЙ ФЕДЕРАЦИИ
ДЕПАРТАМЕНТ НАУЧНО-ТЕХНОЛОГИЧЕСКОЙ ПОЛИТИКИ И ОБРАЗОВАНИЯ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"КРАСНОЯРСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ"

Н.В.Титовская, С.Н.Титовский

МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Учебное пособие

Красноярск 2018

УДК 004.05(075.32)
ББК 32.973-018.2я723

Рецензенты:

И.Н. Коюпченко, канд. физ.-мат. наук, доц. каф. математического моделирования и информационных технологий ТЭИ ФГАОУ ВО СФУ

Постников А.И., канд.техн.наук., доцент каф. Вычислительной техники ИКИТ ФГАОУ ВО СФУ

Титовская Н.В., Титовский С.Н.

Модульное программирование: учеб. пособие / *Н.В.Титовская*; *С.Н.Титовский*, Краснояр. гос. аграр. ун-т. - Красноярск, 2018. - 177 с.

В учебном пособии приведены сведения, необходимые для прохождения учебной практики по получению первичных профессиональных умений и навыков: организация, сроки и место проведения практики; требования к содержанию и оформлению отчета; задания, которые необходимо выполнить во время практики.

Основную часть пособия составляют теоретические материалы, изучение которых необходимо для разработки модульных программ: теоретические основы модульного программирования, структура модуля в языке Pascal, особенности компиляции программ с модульной структурой, а также описание процедур и функций стандартных модулей CRT и GRAPH, что необходимо для разработки программ, составляющих суть заданий на практику.

Также в пособии приведены примеры «каркасных» приложений для разрабатываемых программ.

Учебное пособие предназначено для студентов направления подготовки 38.03.05 (5.38.03.05) «Бизнес-информатика», 09.03.03 (2.09.03.03) «Прикладная информатика».

УДК 004.05(075.32)
ББК 32.973-018.2я723

Содержание

1. Общие сведения

1.1. Цель и задачи практики

Учебная практика по по получению первичных профессиональных умений и навыков является

а) логическим продолжением дисциплин «Информатика и программирование» и «Вычислительные системы, сети и телекоммуникации», изучаемых студентами на первом курсе в течении 1-го и 2-го семестров,

б) основой для успешного изучения последующих дисциплин, изучаемых на втором и третьем курсах связанных с современными методами разработки программных продуктов.

Целью практики является приобретение, закрепление и расширение теоретических знаний и практических навыков разработки программ в областях

- модульного программирования,
- организации экранного (графического) пользовательского интерфейса,
- использования процедур и функций стандартных модулей CRT и GRAPH.

Задачей практики является освоение теоретического материала и разработка на его основе трех программ:

- Программы с модульной структурой,
- Программы с оконным интерфейсом и меню для текстового режима (использующей процедуры и функции модуля CRT),
- Программы с оконным интерфейсом и меню для графического режима (использующей процедуры и функции модуля GRAPH).

1.2. Организация проведения практики

Учебная практика по получению первичных профессиональных умений и навыков проводится после сдачи летней сессии первого курса (второго семестра). По ее окончании студенты, успешно

выполнившие программу практики и защитившие свои программы, получают зачет.

Местом проведения данной практики являются учебные компьютерные классы Института управления и агробизнеса Красноярского государственного аграрного университета.

Для проведения учебной практики и принятия зачетов назначается руководитель из числа преподавателей кафедры Информационных систем и технологий в экономике.

Занятия по учебной практике проводятся ежедневно (с понедельника по субботу включительно) по шесть часов в день в течение всего срока проведения практики, предусмотренного рабочим учебным планом.

Прохождение практики производится в следующем порядке

1. Программы с модульной структурой:
   а) изучение теоретического материала
   б) выполнение практического задания 1
   в) подготовка отчета
2. Графический интерфейс в текстовом режиме:
   а) изучение теоретического материала (основных приемов оформления окон и меню в текстовом режиме, процедур и функций модуля CRT)
   б) выполнение практического задания 2
   в) подготовка отчета
3. Графический интерфейс в графическом режиме:
   а) изучение теоретического материала (основных приемов оформления окон и меню в графическом режиме, процедур и функций модуля GRAPH)
   б) выполнение практического задания 3
   в) подготовка отчета
4. Окончательное оформление отчета и защита у преподавателя

Ориентировочное распределение времени в процентах от общего объема практики приведено в табл. 1.1

Таблица 1.1

Ориентировочное распределение времени

Программы с модульной структурой	**15%**
в том числе:	
изучение теоретического материала	4%

выполнение практического задания 1	9%
подготовка отчета	2%
Графический интерфейс в текстовом режиме	**25%**
в том числе:	
изучение теоретического материала	5%
выполнение практического задания 2	18%
подготовка отчета	2%
Графический интерфейс в графическом режиме	**50%**
в том числе:	
изучение теоретического материала	16%
выполнение практического задания 3	32%
подготовка отчета	2%
Окончательное оформление отчета и защита у преподавателя	**10%**

В процессе прохождения практики преподавателем фиксируется

а) посещение практики

б) выполнение заданий (после выполнения каждого задания студент обязан предъявить его преподавателю)

в) подготовка отчета (после выполнения каждого задания студент обязан подготовить соответствующую часть отчета и показать ее преподавателю)

г) сдача окончательного отчета и защита выполненных заданий у преподавателя.

Студент, не выполнивший программу практики или получивший неудовлетворительную оценку при защите отчета, направляется на практику вторично или отчисляется из университета.

2. Отчет по практике

В процессе прохождения практики и по ее окончании на основании выполненных заданий студент обязан написать отчет. Оформленный отчет сдается преподавателю - руководителю практики.

2.1. Содержание отчета

Отчет по практике должен содержать

- ~ титульный лист (образец приведен в приложении 1)
- ~ цель практики
- ~ задачи практики

Далее следуют три раздела в соответствии с темами выполненных заданий. В каждом разделе приводятся

- ~ тема
- ~ условие задания
- ~ схема межмодульных связей программы
- ~ графические схемы алгоритмов основной программы и всех процедур и функций
- ~ исходные тексты программы и всех модулей, содержащие подробные комментарии
- ~ «скриншоты» с результатами работы программы

После третьего раздела приводится список использованных источников информации, включая ссылки на Internet-ресурсы.

2.2. Основные требования к оформлению отчета

Отчет оформляется на листах белой бумаги формата А4 (210х297 мм) с одной стороны. Шрифт: Times New Roman 14 – 16 (в схемах алгоритмов допускается использование более мелких шрифтов, но не менее 8).

Абзацный отступ: 1,27 см., выравнивание: по ширине (за исключением исходных текстов программы), междустрочный интервал: одинарный – полуторный.

Поля: слева: 20 - 25 мм., справа 5 - 10мм., сверху и снизу: 15 - 20 мм.

Графические схемы алгоритмов должны быть выполнены в соответствии с ГОСТ 19.002-80 и ГОСТ 19.003-80, приведенными в приложении 2 и приложении 3 соответственно

На последней странице основного текста (перед списком использованных источников) должна быть подпись студента – автора отчета.

Отчет должен быть сброшюрован с помощью «скоросшивателя», либо вложен в «файл» (пластиковый пакет для документов формата А4).

К отчету прилагается диск, содержащий электронную версию отчета и файлы с исходными текстами всех программ и модулей, разработанных при выполнении заданий практики.

3. Задания

3.1. Практическое задание 1

Написать программу, имеющую модульную структуру, для рашения пяти задач, приведенных в табл. 3.1. Помимо основной программы в ней должно присутствовать не менее пяти модулей, образующих не менее чем двухуровневую иерархическую структуру.

Основная программа должна в диалоге с пользователем определить номер решаемой задачи и вызвать процедуру решения этой задачи, описанную в одном из модулей.

Процедуры решения задач должны вызавать процедуры/функции ввода исходных данных, собственно обработки и вывода результатов.

Процедуры обработки могут, в свою очередь, вызывать вспомогательные процедуры и функции.

Один из возможных вариантов организации межмодульных связей и распределения процедур и функций по модулям приведен на рис. 3.1.

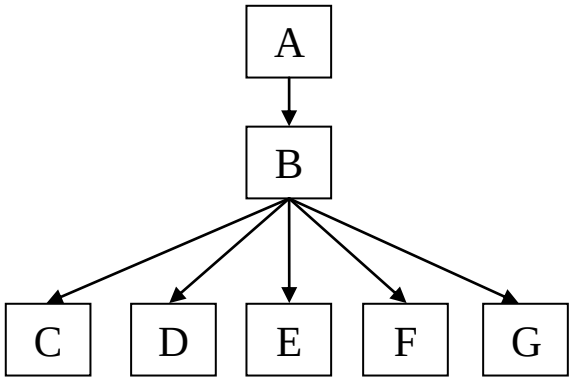


Рис. 3.1. Возможная схема межмодульных связей программы для задания 1

A – основная программа, определяет решаемую задачу и вызывает соответствующую процедуру из модуля B;

B – содержит пять процедур решения задач, вызывающих процедуры/функции из модулей C,D,E,F,G;

C – содержит процедуры/функции ввода исходных данных, собственно обработки и вывода результатов для задачи 1;

D – содержит процедуры/функции ввода исходных данных, собственно обработки и вывода результатов для задачи 2;

E – содержит процедуры/функции ввода исходных данных, собственно обработки и вывода результатов для задачи 3;

F – содержит процедуры/функции ввода исходных данных, собственно обработки и вывода результатов для задачи 4;

G – содержит процедуры/функции ввода исходных данных, собственно обработки и вывода результатов для задачи 5.

При разработке процедур и функций следует руководствоваться принципами структурного и модульного программирования [4, 7, 8-12], которые, помимо всего прочего, запрещают без крайней необходимости использовать глобальные переменные.

Пример «каркасного» приложения для выполнения задания приведен в приложении 4.

3.2. Практическое задание 2

Данное задание основано на практическом задании 1 и предполагает исключение диалога с пользователем при определении номера рещаемой задачи и замену его на оконный пользовательский интерфейс, построенный с использованием возможностей текстового режима работы видеоадаптера. Один из возможных вариантов оформления окна приведен на рис.3.2.

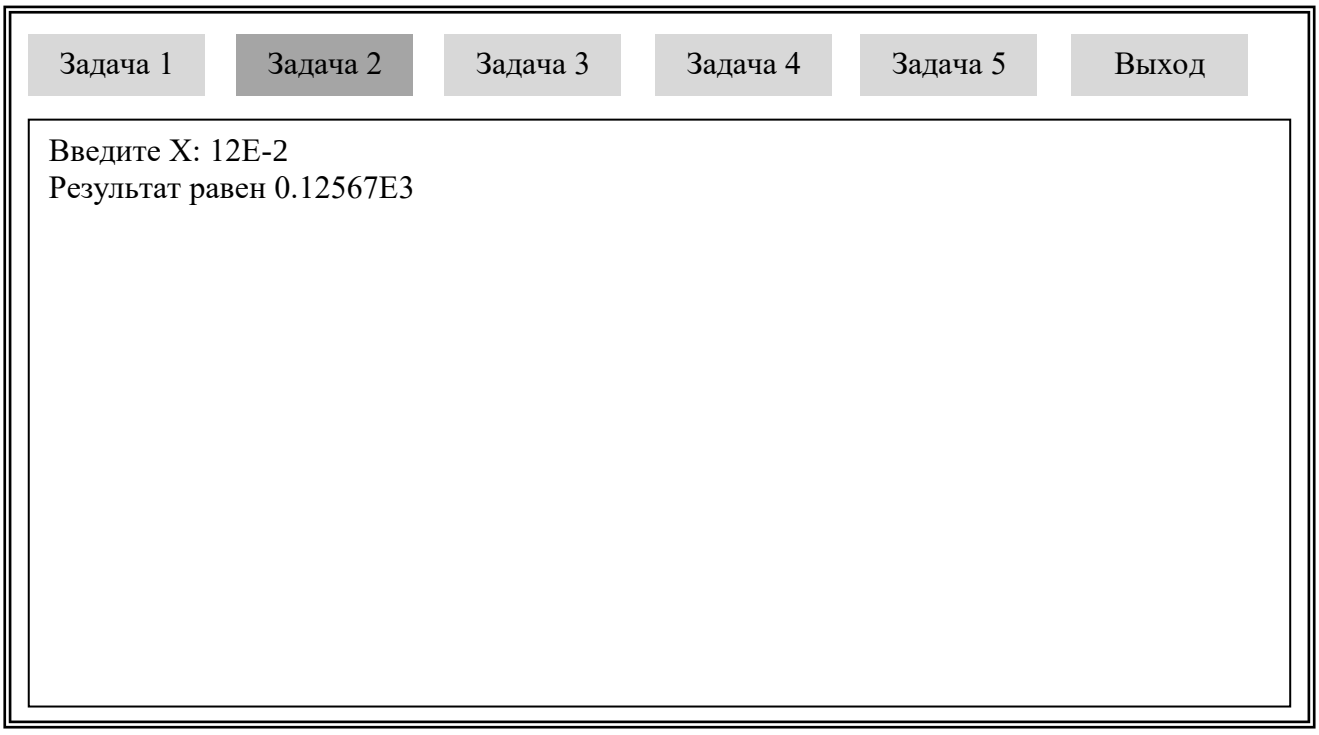


Рис. 3.2. Возможное оформление окна

Приведенное на рисунке окно имеет внешнюю рамку, в верхней части окна расположено меню, позволяющее выбрать решаемую задачу или завершить работу программы. Внутренняя рамка ограничивает клиентскую часть окна, предназначенную для диалога с пользователем во время решения выбранной задачи (ввода исходных данных и вывода сообщений и результатов).

Оформление окна и работу с меню следует организовать с помощью процедур и функций рисования отдельных элементов и выполнения других элементарных операций. На основе этих элементарных процедур и функций строятся более крупные, затем – еще более крупные - и так далее, в соответствии с принципами структурного программирования. Все процедуры и функции, использующиеся при работе с окном и меню, должны быть оформлены в виде одного или нескольких модулей.

В отличие от предыдущего задания в данном случае основная программа должна вызывать процедуру оформления окна, далее в цикле с помощью функции работы с меню определять номер задачи, выбранной пользователем и вызывать соответствующую процедуру до тех пор, пока не будет выбран пункт меню завершения программы. В этом случае экран должен очищаться, а программа – завершаться.

Диалог с пользователем во время решения выбранной задачи должен происходить в рамках клиентской части окна, т. е. общее оформление окна не должно нарушаться. Для этого перед вызовом процедуры решения задачи следует организовать окно в пределах, ограниченных внутренней рамкой.

Пример «каркасного» приложения для выполнения задания приведен в приложении 5.

Таблица 3.1.

Варианты заданий 1

Вариант	Задача 1	Задача 2	Задача 3
1	$z = \sqrt{\left\|\frac{\sin x}{(x^2+4)}\right\|} + \sin\left\|a^4\right\|$, $a = \sin x$	$y = \ln\left(\sqrt{\frac{\sin x}{5x+4}}\right) - e^{\frac{a}{x}} + \sqrt{\sin\frac{x}{x^2+1}}$, $a = 0{,}5$	Вычислить сумму ряда $S = \frac{1}{x} + \frac{1}{x^2} + \ldots + \frac{1}{x^n}$
2	$y = \sin x + \left\|x^2+1\right\|^5 - \sqrt{\left\|\frac{x^2}{x^2+5}\right\|}$	$y = \ln\left(\sin x \times \sqrt{\frac{1+\cos\frac{a}{x}}{a \times x \times \sin(ax)}}\right) + \sqrt{\frac{\sin x}{x}}$, $a = 54 \times 10^{-3}$	Вычислить сумму ряда $S = \sum_{i=1}^{\infty} \frac{1}{i^2}$ с погрешностью $E>0$
3	$y = \left\|x^4+1\right\|^7 + \sin x^2 + 5z$, $z = \sqrt{\left(\sin x^4\right)}$	$y = \begin{cases} \sqrt{\ln x}, & x > 0 \\ 2x & x = 0 \\ e^{x^2} & x < 0 \end{cases}$	Вычислить произведение ряда $P = \left(1+\frac{1}{1^2}\right)\left(1+\frac{1}{2^2}\right) \times \ldots \times \left(1+\frac{1}{n^2}\right)$
4	$z = \frac{\cos x}{(\sin x)^2 + 5} - e^{x^2} + 4{,}48 \times 10^{-1}$	$y = \frac{\sqrt{ax}}{\sin(\ln x)} - \sqrt{\frac{\cos x^2}{x^2}}$, $a = 10$	Вычислить сумму ряда $S = \sum_{i=1}^{\infty} \frac{1}{i(i+1)}$ с погрешностью $E > 0$
5	$z = \sqrt{\left\|\cos^2 x + 1\right\|} + e^{\sin x} - 0{,}36 \times 10^2$	$y = \begin{cases} tg(2a + a/x), & x > 0 \\ \pi & x = 0 \\ \sin x & x < 0 \end{cases}$ $a = 0.64 \cdot 10^{-2}$	Вычислить сумму ряда $S = \frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + \ldots + \frac{1}{\sin 1 + \ldots + \sin n}$

6	$y = e^{x^2} + \cos(2x + 5) + \sqrt{\lvert x^2 + 5 \rvert}$	$y = e^{\left(\frac{a}{x}+4\right)} + \sqrt{\frac{\ln(\sin x)}{\cos ax}} + (\sin 2x + 5x)^3$, a = 0,25	Вычислить сумму ряда $S = \sum_{i=1}^{\infty} \frac{(-1)^i}{5^2}$ с погрешностью E > 0
7	$y = e^{x^4} + \frac{\sin x}{\cos(x+2)^4 + 10} + \ln(x^2 + 5)$	$y = \sqrt[4]{\frac{x}{a} + \frac{a}{x}} + \ln\frac{\sin x}{\cos 2x}$; a = 0,4	Вычислить сумму ряда $S = 1 + \frac{x}{1} + \frac{x^2}{2} + \ldots + \frac{x^n}{n}$
8	$y = e^{\lvert x^2 + 2 \rvert^3} + \cos^2 x + \sqrt{x^2 + 1} + 1{,}15$	$y = \begin{cases} x^2/a, & -2 \le x \le 2; \\ \sqrt{x}, & x > 2 \\ x^3, & x < -2; \end{cases}$; $a = 0.57 \cdot 10^2$	Вычислить сумму ряда $S = \sum_{i=1}^{\infty} \frac{(-2)^i}{3^i}$ с погрешностью E > 0
9	$z = \frac{\lvert \cos x \rvert^5 + \frac{\sin x}{\lvert x^2 + 1 \rvert^3} + 10y}{\ln \lvert x^2 + 1 \rvert}$, y =	$y = \left(\frac{1}{2\pi x}\right)^{\frac{1}{2}} \times \sqrt[3]{\frac{\ln(\sin x)}{(\sin 2x + \cos x)}}$	Вычислить сумму S = $x^{(1^2)} + x^{(2^2)} + x^{(3^3)} + \ldots + x^{(n^2)}$
10	$z = e^{x^3} + \frac{\cos x}{\sin^2(x+1) + 5 \times 10^{-1}} - \frac{}{0{,}0012 \times 10^4}$	$y = x^a \times \ln x + \frac{x^a \times e^{ax}}{\sin x \times \cos x}$; a = $2{,}34 \times 10^{-2}$	Вычислить сумму $S = \sum_{i=1}^{\infty} \frac{(-1)^{i+1}}{i(i+1)(i+2)}$ с погрешностью E > 0
11	$y = (\sin x)^2 + e^{\lvert x^2 + 1 \rvert^5} - \ln(x^2 + 1)$	$y = \frac{\sin x^2}{\cos x^2 + 5} + \sqrt[3]{\frac{\ln(x-2)}{\sin x^2 + x^2 + 1}}$	Вычислить сумму S = $x^{(2^1)} + x^{(2^2)} + \ldots + x^{(2^n)}$ с погрешностью E > 0

12	$y = e^{x^3} + \sin^2 x + \frac{\cos x}{\sin^2(x)+1} - 0{,}14\times10^{-1}$	$y = \sqrt{\frac{\sin x \times x^2}{\cos x + \sin x^2}} - \ln\left(x + \sqrt{\frac{\sin x}{\cos 2x}} - \ln x\right)$	Вычислить сумму $S = \sum_{i=0}^{\infty} \frac{1}{4^i + 5^{i+2}}$ с погрешностью $E > 0$
13	$y = \ln\left\|x^4+1\right\| + \sqrt[5]{\left\|x^2+10\right\|} + 1{,}64\times10^2$	$y = \sqrt{e^{bx}} + \text{arctg}\left(\frac{x+2}{2}\right) + \frac{\pi(1+x)x^a}{\sin x}$; a = 3,5; b = 1,44	Вычислить сумму S = sin x + sin(sin x)+ …+ sin(sin(..sin x)…)
14	$y = 3\sin x + \sqrt{\left\|x^2+5\right\|} + x^2 - 0{,}5\times10^{-2}$	$y = \left(\frac{x^{a-1}}{1+x} + \sqrt{x^a \times (1+x)^3}\right) / \ln(1+x)$; a = 16×10^{-1}	Вычислить сумму $S = \sum_{k=1}^{\infty} \frac{\sin k}{5^k}$ с погрешностью $E > 0$
15	$y = \sqrt{\left\|x^4+1\right\|} + \cos\left(x^2 + \frac{\pi}{4}\right) - \ln\left\|x^2+4\right\|$	$y = \left(\frac{\sin x}{\sqrt{x}} + \frac{\sin^2 ax}{x^2}\right)^{1/2} + \sqrt{x + \frac{\pi}{2} \times e^{-\left\|a\right\|}}$; a = $0{,}0034\times10^3$	Вычислить сумму S = (x+h)+2(x+2h)+3(x+3h)+…+N(x+Nh)
16	$y = \ln\left\|x^4 + \sqrt{\left\|x^2+6\right\|}\right\| + (x+5)^2$	$y = \sqrt{\frac{\ln x}{x-1}} + \left(\frac{e^{-ax^2} + 4x}{\sin x}\right)$; a = 0,74	Вычислить сумму $S = \sum_{n=3}^{\infty} \frac{1}{n \times \ln n (\ln(\ln n))^2}$ с заданной погрешностью $E > 0$
17	$y = \text{arctg}(x+1) + e^{x+1} + \ln(x^2+2)$	$z = \ln\left(\left(y - \sqrt{x}\right)\left(x - \frac{y}{y+x^2}\right)\right)^{\frac{1}{2}}$; y = 10	Вычислить сумму первых N слагаемых $S = 1 + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \frac{9}{5} + \ldots$

18	z = 2 $\cos 2\left(x-\frac{\pi}{6}\right)+arctg\left(x-\frac{\pi}{6}\right)+\left\|x^2+1\right\|^5$	$z=\ln\left(\cos^2(x+2)+\sin x\right)+\sqrt[4]{\frac{\sin x}{x}}$	Вычислить сумму ряда S = $\sum_{k=1}^{\infty}\frac{k}{(\sin k)^k+4k^5}$ с заданной погрешностью E > 0
19	$z=\left\|x^2+4\right\|^7+\sqrt{x^2+4}$ - ln (x^2 +4)	$z=\sqrt{x^4-64}\times\left(\frac{\ln(y)+\cos(x^2)}{4x+5}\right)^{1/3}$; y = 1,2	Найти сумму первых N слагаемых S = $\frac{1}{2}+\frac{3}{4}+\frac{5}{6}+\frac{7}{8}+....$
20	$y=\sqrt{\left\|\sin^2(x+1)+5\right\|}+\left\|x^4+1\right\|^5$	$z=x^{y^a}+\sqrt{xy}-\frac{\ln(x+y)}{(2x-y)^{1/2}}$; y = 6,8 ; a = 0,15	Вычислить сумму S = $\sum_{n=1}^{\infty}\frac{1}{n^2+\sin n}$ с погрешностью E > 0
21	$y=e^{-2x}+\ln(\sqrt{x^2+4})+14{,}48\times10^{-5}$	$z=\left(x^4+6\right)^{1/3}\times\frac{y+\sqrt{\cos\left(x^2+\pi/4\right)}}{\ln(x)-2x}$; $y=0{,}72\times10^{-2}$	Вычислить произведение P = $\prod_{i=1}^{10}(2+\frac{1}{i})$
22	$y=e^{\left\|\cos x+\sin x\right\|}\times\ln\left\|x^2+1\right\|+\left\|x^2+1\right\|^6$	$z=\frac{e^{-x^2}}{x}+\left(\frac{\ln(4x)}{\sin x}\right)^{1/3}$	Вычислить произведение P = $\prod_{k=1}^{n}(\frac{k}{k+1}-\cos^k x)$
23	$z=\left\|x^4+12\right\|^5+\sqrt{\sin^2 x+\cos^2 x+4}$	$y=\left(\frac{x+5}{\sin x^3}\right)^{1/2}+\sqrt[3]{\ln\frac{x^2+2x}{x}}$	Вычислить сумму S = $\sum_{i=1}^{12}\frac{i}{\frac{1}{2}+\frac{1}{3}+...+\frac{1}{i+1}}$

24	$z = \left\|2x^2+5\right\|^3 + \sqrt{x^4+2}$	$y = \frac{\sqrt{\sin x \times 2x}}{2x} + \frac{\ln\left(tg\sqrt{\frac{2x}{\sin(\ln x)}}\right)}{(x+4)^{1/3}}$	Вычислить сумму $S = \sum_{k=1}^{\infty} \frac{1}{(2k+1)^2}$ с погрешностью E > 0
25	$z = \left\|2x^2+5\right\|^3 + \sqrt{x^4+2} - 0{,}024 \times 10^2$	$y = \frac{(4x+\sin^2 x)^{1/2}}{\ln(\sqrt{x})-10} + \frac{\ln(\sin x)}{x}$	Вычислить сумму $S = \sum_{i=1}^{10} \frac{x+\cos(ix)}{2^i}$

Таблица 3.1.
Варианты заданий 1
Продолжение

Вариант	Задача 4	Задача 5
1	Написать программу расчета среднего арифметического (СА) значения положительных элементов в одномерном массиве, имеющих четные индексы.	В квадратной матрице [Aij], i,j=$\overline{1,M}$ заменить нулями элементы с четной суммой индексов, не превышающие некоторого числа X.
2	Написать программу вычисления суммы отрицательных, произведения положительных и количества нулевых значений в одномерном массиве.	Получить матрицу [Bij], i,j=$\overline{1,M}$ из матрицы [Aij], i,j=1 путем перестановки столбцов - первого с последним, второго с предпоследним и т.д.
3	Написать программу расчета суммы положительных элементов одномерного массива, имеющих нечетные индексы.	Получить новую матрицу [Bi,j], i,j=$\overline{1,M}$ из матрицы[Aij], путем перестановки сегментов по диагонали.
4	Упорядочить одномерный массив в порядке неубывания.	Получить новую матрицу [Bi,j], i,j=$\overline{1,M}$ из матрицы[Aij], путем перестановки сегментов по часовой стрелке.

5	Написать программу расчета СА отрицательных элементов в одномерном массиве. Заменить минимальный элемент в одномерном массиве на СА.	В произвольной матрице [Aij] i=$\overline{1,M}$; j=$\overline{1,N}$ найти минимальный и максимальный элементы, указать номера строк и номера столбцов, на пересечении которых они находятся.
6	Упорядочить одномерный массив в порядке невозрастания.	Из одномерного массива [Xi] i=$\overline{1,64}$ получить действительную квадратную матрицу 8*8, элементами которой являются числа $x_1,\ldots,x_{64}$ расположенные в ней по столбцам.
7	В одномерном массиве поменять местами максимальный и минимальный элементы.	Из одномерного массива [Xi] i=$\overline{1,64}$ получить действительную квадратную матрицу 8*8, элементами которой являются числа $x_1,\ldots,x_{64}$ расположенные в ней по строкам.
8	Написать программу расчета среднего геометрического (СГ) положительных элементов в одномерном массиве. Заменить максимальный элемент в одномерном массиве на СГ.	В произвольной матрице [Aij] i=$\overline{1,M}$; j=$\overline{1,N}$ столбец, содержащий максимальный элемент, заменить на сумму всех элементов матрицы.
9	Произвести попарные перестановки элементов одномерного массива: первый элемент поменять местами с последним, второй элемент – с предпоследним и т.д.	Получить квадратную матрицу [Aij], i,j=$\overline{1,M}$ элементы главной диагонали которой будут числа в диапазоне от 1 до N.
10	Отыскать последний положительный элемент в одномерном массиве и заменить его на СА элементов массива.	Получить квадратную матрицу [Aij], i,j=$\overline{1,M}$ элементы главной диагонали которой будут числа в диапазоне от N до 1.
11	Дан одномерный массив [Ai]. Сформировать одномерный [Bi] массив из элементов массива [Ai] по закону $B_i = \sum_{j=0}^{N}$ Ai+j, i = $\overline{1,M}$; N = M - i	Получить квадратную матрицу [Aij], i,j=$\overline{1,M}$ элементы побочной диагонали которой будут числа, получающиеся в результате перемножения i*(i+1), где I – номер строки.

12	Из одномерного массива [Ai] сформировать одномерный массив [Bi], записав в него сначала элементы массива A, имеющие четные индексы, потом – элементы с нечетными индексами.	Получить квадратную матрицу [Aij], i,j=$\overline{1,M}$ элементы которой будут получены следующим образом: в каждой строке матрицы первые (N-i+1)элементов заполнены номером строки, остальные – нулями.
13	Отыскать последний отрицательный элемент в одномерном массиве и заменить его на СГ элементов массива.	Получить матрицу [Cij] i,j= $\overline{1,M}$ из матриц [Aij] i,j=$\overline{1,M}$ и [Bij] i,j=$\overline{1,M}$ путем умножения элементов каждой строки матрицы [Aij] на максимальный элемент соответствующей строки матрицы [Bij].
14	Заменить в одномерном массиве нулевые элементы на значение минимального элемента.	В квадратной матрице [Aij] i,j=$\overline{1,M}$ среди элементов расположенных ниже главной диагонали найти минимальный, а среди элементов выше главной диагонали – максимальный. Вывести координаты этих элементов.
15	Сформировать массив [Xi], элементы которого равны частоте встречаемости элементов массива [Bi] среди элементов массива [Ai] Определить, какой элемент массива [Bi] чаще всего встречается в [Ai].	В квадратной матрице [Aij] i,j=$\overline{1,M}$ среди элементов, расположенных ниже побочной диагонали, определить количество положительных элементов, а среди элементов выше побочной диагонали – количество отрицательных элементов.
16	Сформировать массив [Xi], элементы которого равны полусумме двух соседних элементов одномерного массива [Yi].	Из квадратной матрицы [Aij] i,j=$\overline{1,M}$ сформировать одномерный массив [Xi] i=$\overline{1,2M}$ по следующему правилу: элементами одномерного массива [Xi] с нечетными индексами будут элементы главной диагонали [Aij], с четными – побочной диагонали [Aij].

17	Сформировать массив [Ai] из элементов одномерного массива [Bi] по закону $A_i = (B_i + B_{N-i+1})/4$, $i = \overline{1,N}$	Сформировать одномерный массив [Xi] $i=\overline{1,M}$ из сумм положительных элементов строк матрицы [Aij] $j,i=\overline{1,M}$, попутно определяя номера строк матрицы [Aij] $i,j=\overline{1,M}$, в которых отсутствуют положительные элементы.
18	Сформировать массив [Ai] из элементов одномерного массива $[B_i]$ $j = \overline{1,N}$ по закону $A_i = B_i + B_{N/2+i}$; $i = \overline{1,\frac{N}{2}}$	Сформировать одномерный массив [Bi] $i=\overline{1,M}$ из минимальных элементов строк прямоугольной матрицы[Aij] $i=\overline{1,M}$, $j=\overline{1,N}$. Подсчитать количество элементов массива [Bi] попавших в интервал (x,y).
19	Из одномерного массива [Bi] сформировать массив [Ai] по закону $A_j = (\sum_{i=1}^{j+1} B_i)/(j+1)$; $j = \overline{1,N-1}$	Сформировать одномерный массив [Bi] $i=\overline{1,M}$ из максимальных элементов столбцов прямоугольной матрицы [Aij] $i=\overline{1,M}$, $j=\overline{1,N}$. В массиве [Bi] поменять местами первый отрицательный и последний положительный элементы.
20	Из одномерного массива [Bi] сформировать массив [Xi] по следующему закону: $Xi = \begin{cases} 1 & B_i > Y, \\ 0 & B_i = Y, \\ -1 & B_i < Y, \end{cases}$ где y – некоторая константа	В квадратной матрице [Aij] $i,j=\overline{1,M}$ заменить элементы главной и побочной диагоналей на минимальный элемент главной диагонали.
21	В одномерном массиве переставить местами соседние элементы с четными и нечетными индексами.	В произвольной матрице [Aij] $i=\overline{1,M}$, $j=\overline{1,N}$ поменять местами строку, содержащую минимальный элемент, со строкой, содержащей максимальный элемент.

22	В одномерном массиве вычислить сумму элементов, значения которых кратны некоторому значению X.	В квадратной матрице [Aij] i,j=$\overline{1,M}$, найти максимальный элемент среди элементов, стоящих на главной и побочной диагоналях, и поменять его местами с элементом, стоящим на пересечении этих диагоналей.
23	В одномерном массиве подсчитать количество элементов, значениями которых являются простые числа (простое число – это число, делящееся нацело только на единицу и само на себя).	В квадратной матрице [Aij] i,j=$\overline{1,M}$ определить номер столбца матрицы, имеющего наибольшую сумму элементов. Поменять этот столбец со строкой имеющей наименьшую сумму элементов.
24	Сформировать массив [Bi], содержащий последовательность чисел Фибоначчи: $B_i = B_{i-1} + B_{i-2}$; i = $\overline{3,N}$; $B_1 = X_1$, $B_2 = X_2$ где X_1, X_2 - некоторые числа.	В квадратной матрице [Aij] i,j=$\overline{1,M}$ найти наибольшее из значений элементов, расположенных в первом и третьем секторах матрицы, полученных в результате пересечения главной и побочной диагонали.
25	Вычислить сумму правых разностей элементов одномерного массива [Bi] $S = \sum_{i=1}^{N-1} (B_i - B_{i+1})$	В квадратной матрице [Aij] i,j=$\overline{1,M}$ найти наибольшее из значений элементов, расположенных во втором и четвертом секторах матрицы, полученных в результате пересечения главной и побочной диагонали.

3.3. Практическое задание 3

Написать программу, выполняющую четыре операции над графическим объектом («фигурой»):движение, вращение, управление движением, управление вращением.

Вид фигуры зависит от варианта и приведен в табл. 3.2.

Движение – перемещение фигуры в одном из восьми направлений пока нажата соответствующая клавиша на клавиатуре.

Вращение – поворот фигуры вокруг своего геометрического центра вправо (по часовой стрелке) или влево (против часовой стрелки) пока нажата соответствующая клавиша на клавиатуре.

Управление движением – самостоятельное перемещение фигуры в одном из восьми направлений, задаваемых однократным нажатием соответствующих клавиш на клавиатуре.

Управление вращением – самостоятельный поворот фигуры вокруг своего геометрического центра вправо или влево в соответствии с однократным нажатием соответствующих клавиш на клавиатуре.

Программа должна иметь оконный интерфейс, аналогичный изображенному на рис 3.2., и модульную структуру, соответствующую требованиям задания 1.

Один из возможных вариантов организации межмодульных связей и распределения процедур и функций по модулям приведен на рис. 3.3.

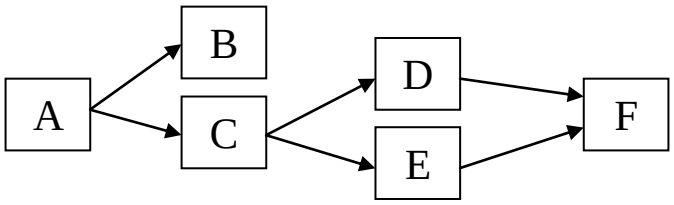


Рис. 3.3. Возможная схема межмодульных связей программы для задания 3

A – основная программа, оформляет окно и определяет выполняемую операцию путем вызыва соответствующих процедур/функций из модуля B, для выполнения операции вызывает одну из процедур модуля C;

B – содержит процедуры/функции оформления окна и работы с меню, разработанные в соответствии с требованиями к аналогичным процедурам, изложенными в задании 2;

C – содержит четыре процедуры выполнения операций, вызывающих процедуры/функции из модулей D и E;

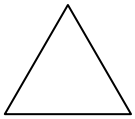
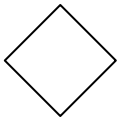
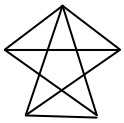
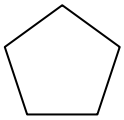
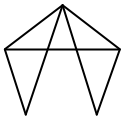
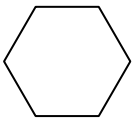
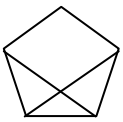
D – содержит восемь процедур перемещения фигуры на один шаг, использующих процедуры модуля F;

E – содержит процедуры поворота фигуры на элементарный угол, использующих процедуры модуля F;

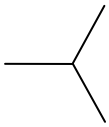
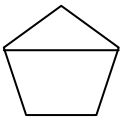
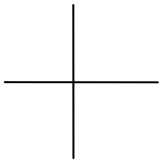
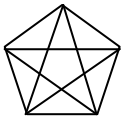
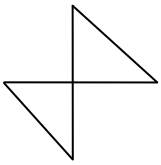
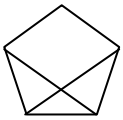
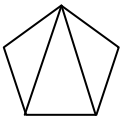
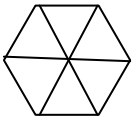
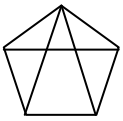
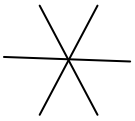
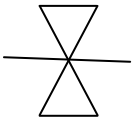
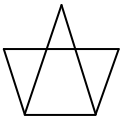
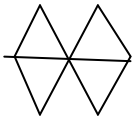
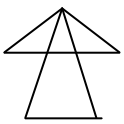
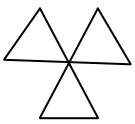
F – содержит процедуры рисования и стирания фигуры;

Пример «каркасного» приложения для выполнения задания приведен в приложении 6.

Таблица 3.2.
Варианты заданий 2

Вариант	Фигура[1]	Вариант	Фигура
1		14	
2		15	
3		16	
4		17	

[1] Все фигуры состоят из линий, соединяющих вершины правильного многоугольника, вписанного в окружность, и геометрическим центром фигуры считается центр этой окружности

5		18	
6		19	
7		20	
8		21	
9		22	
10		23	
11		24	
12		25	
13			

4. Теоретические сведения[2]

4.1. Модули

Стандартный Паскаль не предусматривает механизмов раздельной компиляции частей программы с последующей их сборкой перед выполнением. Более того, последовательное проведение в жизнь принципа обязательного описания любого объекта перед его использованием делает фактически невозможным разработку разнообразных библиотек прикладных программ [1-3]. Точнее, такие библиотеки в рамках стандартного Паскаля могут существовать только в виде исходных текстов и программист должен сам включать в программу подчас весьма обширные тексты различных поддерживающих процедур, таких, как процедуры матричной алгебры, численного интегрирования, математической статистики и т.п.

Вполне понятно поэтому стремление разработчиков коммерческих компиляторов Паскаля включать в язык средства, повышающие его модульность. Чаще всего таким средством является разрешение использовать внешние процедуры и функции, тело которых заменяется стандартной директивой EXTERNAL. Разработчики Турбо Паскаля пошли в этом направлении еще дальше, включив в язык механизм так называемых модулей.

Модуль - это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части[1-3, 13, 16]. По своей организации и характеру использования в программе модули Турбо Паскаля близки к модулям-пакетам (PACKAGE) языка программирования Ада. В них так же, как в пакетах Ады, явным образом выделяется некоторая «видимая» интерфейсная часть, в которой сконцентрированы описания глобальных типов, констант и переменных, а также приводятся заголовки глобальных процедур и функций. Появление объектов в интерфейсной части делает их доступными для других модулей и основной программы. Тела

[2] В данном разделе использован материал с pascal.kansoftware.ru

процедур и функций располагаются в исполняемой части модуля, которая может быть скрыта от пользователя.

Насколько сильно изменяются свойства языка Паскаль при введении механизма модулей, свидетельствует следующее замечание его автора Н. Вирта, сделанное им по поводу более позднего языка Модула-2: «Модули - самая важная черта, отличающая язык Модула-2 от его предшественника Паскаля»[5].

Модули представляют собой прекрасный инструмент для разработки библиотек прикладных программ и мощное средство модульного программирования. Важная особенность модулей заключается в том, что компилятор Турбо Паскаля размещает их программный код в отдельном сегменте памяти. Максимальная длина сегмента не может превышать 64 Кбайта, однако количество одновременно используемых модулей ограничивается лишь доступной памятью, что дает возможность создавать весьма крупные программы.

4.1.1. Структура модулей

Модуль имеет следующую структуру:

UNIT <имя>;
INTERFACE
<интерфейсная часть>
IMPLEMENTATION
<исполняемая часть>
BEGIN
<инициирующая часть>
END.

Здесь UNIT - зарезервированное слово (единица); начинает заголовок модуля; <имя> - имя модуля (правильный идентификатор); INTERFACE - зарезервированное слово (интерфейс); начинает интерфейсную часть модуля;

IMPLEMENTATION - зарезервированное слово (выполнение); начинает исполняемую часть;

BEGIN - зарезервированное слово; начинает инициирующую часть модуля; конструкция BEGIN <Инициирующая часть> необязательна;

END - зарезервированное слово - признак конца модуля.

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

4.1.2. 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 Турбо Паскаля и возможности подключения средств, облегчающих разработку крупных программ, это имя должно совпадать с именем дискового файла, в который помещается исходный текст модуля[1-3]. Если, например, имеем 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 CRT, Graph, Global;

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).

4.1.3. 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[1-3, 13-16]. При объявлении глобальных подпрограмм в интерфейсной части указывается только их заголовок, 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);
```

Если теперь в основной программе написать предложение

```
Uses Cmplx;
```

то в программе станут доступными тип COMPLEX и две процедуры - ADDC и MULC из мдуля CMPLX

Отметим, что объявление подпрограмм в интерфейсной части автоматически сопровождается их компиляцией с использованием дальней модели памяти. Таким образом обеспечивается доступ к подпрограммам из основной программы и других модулей. Следует учесть, что все константы и переменные, объявленные в интерфейсной части модуля, равно как и глобальные константы и переменные основной программы, помещаются компилятором Турбо Паскаля в общий сегмент данных (максимальная длина сегмента 65536 байт). Порядок появления различных разделов объявлений и их количество может быть произвольным. Если в интерфейсной части объявляются внешние подпрограммы или подпрограммы в машинных кодах, их тела (т.е. зарезервированное слово EXTERNAL, в первом случае, и машинные коды вместе со словом INLINE - во втором) должны следовать сразу за их заголовками в исполняемой части модуля (не в интерфейсной!). В интерфейсной части модулей нельзя использовать опережающее описание.

4.1.4. 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 инициирующей части[1-3, 13-16].

Описанию подпрограммы, объявленной в интерфейсной части модуля, в исполняемой части должен предшествовать заголовок, в котором можно опускать список формальных переменных (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);
Implementation
Procedure AddC;
begin
z.re := x.re +Y.re;
z.im := x.im +y.im
end;
end.
```

Локальные переменные и константы, а также все программные коды, порожденные при компиляции модуля, помещаются в общий сегмент памяти.

4.1.5. Инициирующая часть

Инициирующая часть завершает модуль. Она может отсутствовать вместе с начинающим ее словом BEGIN или быть пустой - тогда за BEGIN сразу следует признак конца модуля (слово END и следующая за ним точка) [1-3, 13-16].

В инициирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти операторы исполняются до передачи управления основной программе и обычно используются для подготовки ее работы. Например, в них могут инициироваться переменные, открываться нужные файлы,

устанавливаться связи с другими ПК по коммуникационным каналам и т.п.:

```
Unit FileText;
Interface
Procedure Print(s : string);
Implementation
var
f: text; const
name = 'output.txt'; Procedure Print;
begin
WriteLn(f, s)
end;
{ Начало инициирующей части: }
begin
assign(f, name);
rewrite(f);
{ Конец инициирующей части }
end.
```

Не рекомендуется делать инициирующую часть пустой, лучше ее опустить: пустая часть содержит пустой оператор, которому будет передано управление при запуске программы. Это часто вызывает проблемы при разработке оверлейных программ (см. гл.11).

4.1.6. Компиляция модулей

В среде Турбо Паскаля имеются средства, управляющие способом компиляции модулей и облегчающие разработку крупных программных проектов. В частности, определены три режима компиляции: COMPILE, MAKE и BUILD. Режимы отличаются только способом связи компилируемого модуля или основной программы с другими модулями, объявленными в предложении USES,

При компиляции модуля или основной программы в режиме COMPILE все упоминающиеся в предложении USES модули должны быть предварительно откомпилированы, и результаты компиляции помещены в одноименные файлы с расширением TPU. Например, если в программе (модуле) имеется предложение

```
Uses Global;
```

то на диске в каталоге, объявленном опцией UNIT DIRECTORIES, уже должен находиться файл GLOBAL.TPU. Файл с расширением TPU (от англ. Turbo Pascal Unit) создается автоматически в результате компиляции модуля (если основная программа может компилироваться без создания исполняемого EXE-файла, то компиляция модуля всегда приводит к созданию TPU-файла).

В режиме MAKE компилятор проверяет наличие TPU-файлов для каждого объявленного модуля. Если какой-либо из файлов не обнаружен, система пытается отыскать одноименный файл с расширением PAS, т.е. файл с исходным текстом модуля, и, если искомый файл найден, приступает к его компиляции. Кроме того, в этом режиме система следит за возможными изменениями исходного текста любого используемого модуля. Если в PAS-файл (исходный текст модуля) внесены какие-либо изменения, то независимо от того, есть ли уже в каталоге соответствующий TPU-файл или нет, система осуществляет его компиляцию перед компиляцией основной программы. Более того, если изменения внесены в интерфейсную часть модуля, то будут перекомпилированы также и все другие модули, обращающиеся к нему. Режим MAKE, таким образом, существенно облегчает процесс разработки крупных программ с множеством модулей: программист избавляется от необходимости следить за соответствием существующих TPU-файлов их исходному тексту, так как система делает это автоматически.

В режиме BUILD существующие TPU-файлы игнорируются, и система пытается отыскать (и компилировать) соответствующий PAS-файл для каждого объявленного в предложении USES модуля. После компиляции в режиме BUILD программист может быть уверен в том, что учтены все сделанные им изменения в любом из модулей.

Подключение модулей к основной программе и их возможная компиляция осуществляются в порядке их объявления в предложении USES. При переходе к очередному модулю система предварительно отыскивает все модули, на которые он ссылается. Ссылки модулей друг на друга могут образовывать древовидную структуру любой сложности, однако запрещается явное или косвенное обращение модуля к самому себе. Например, недопустимы следующие объявления:

```
Unit A;          Unit B;
```

```
Interface          Interface
Uses B;            Uses A;
.......            .......
Implementation     Implementation
.......            .......
end.               end.
```

Это ограничение можно обойти, если «спрятать» предложение USES в исполняемые части зависимых модулей:

```
Unit A;            Unit B;
Interface          Interface
.......            .......
Implementation     Implementation
Uses B;            Uses A;
......             .......
end.               end.
```

Дело в том, что Турбо Паскаль разрешает ссылки на частично откомпилированные модули, что приблизительно соответствует опережающему описанию подпрограммы. Если интерфейсные части любых двух модулей независимы (это непременное условие!), Турбо Паскаль сможет идентифицировать все глобальные идентификаторы в каждом из модулей, после чего откомпилирует тела модулей обычным способом.

4.1.7. Доступ к объявленным в модуле объектам

Пусть, например, мы создаем модуль, реализующий арифметику комплексных чисел (такая арифметика ни в стандартном Паскале, ни в Турбо Паскале не предусмотрена). К сожалению, в Турбо Паскале нельзя использовать функции, значения которых имели бы структурированный тип (запись, например), поэтому арифметика комплексных чисел реализуется четырьмя процедурами:

```
UNIT 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; begin
    z.re := x.re + y.re; z . im := x.im + y. im
    end {AddC};
    Procedure SubC;
    begin
    z.re := x.re - y. re ;
    z.im := x.im - y.im
    end {SubC};
    Procedure MulC;
    begin
    z.re := x.re*y.re - x.im*y. im;
    z.im := x.re*y.im + x.im*y.re
    end {MulC};
    Procedure DivC;
    var
    zz : real;
    begin
    zz := sqr(y.re) + sqr(y.im);
    z. re := (x.re * y.re + x.im * y.im) / zz;
    z.im := (x.re * y.im - x.im * y.re) / zz
    end {DivC};
    end.
```

Текст этого модуля следует поместить в файл CMPLX.PAS. Вы можете его откомпилировать, создав TPU-файл, после чего Вашей программе станут доступны процедуры из новой библиотеки. Например, в следующей программе осуществляются четыре арифметические операции над парой комплексных чисел.

```
Uses Cmplx;
var
a, Ь, c : complex;
begin
a.re := 1; a.im := 1;
b.re := 1; b.im := 2;
AddC(a, b, c);
WriteLn("Сложение: "c.re:5:1, c.im:5:1,"i") ;
SubC(a, b, c) ;
WriteLn("Вычитание: "c.re:5:1, c.im:5:1,"i");
MulC(a, b, c);
WriteLn("Умножение: "c.re:5:1, c.im:5:l,"i") ;
DivC(a, b, c);
WriteLn("Деление: "c.re:5:l, c.im:5:1,"i");
end.
```

После объявления Uses Cmplx программе стали доступны все объекты, объявленные в интерфейсной части модуля CMPLX. При необходимости можно переопределить любой их этих объектов, как это произошло, например, с объявленной в модуле типизированной константой C. Переопределение объекта означает, что вновь объявленный объект «закрывает» ранее определенный в модуле одноименный объект. Чтобы получить доступ к «закрытому» объекту, нужно воспользоваться составным именем: перед именем объекта поставить имя модуля и точку. Например, оператор

```
WriteLn(cmplx.c.re:5:l, cmplx.c.im:5:1,"i");
```

выведет на экран содержимое «закрытой» типизированной константы из предыдущего примера.

4.1.8. Стандартные модули

В Турбо Паскале имеется восемь стандартных модулей, в которых содержится большое число разнообразных типов, констант, процедур и функций [1-3, 5]. Этими модулями являются SYSTEM, DOS, CRT, PRINTER, GRAPH, OVERLAY, TURBO3 и GRAPH3. Модули GRAPH, TURBO3 и GRAPH3 выделены в отдельные TPU-файлы, а остальные входят в состав библиотечного файла TURBO.TPL. Лишь один модуль SYSTEM подключается к любой

программе автоматически, все остальные становятся доступны только после указания их имен в списке, следующем за словом USES.

Ниже приводится краткая характеристика стандартных модулей.

Модуль SYSTEM. В него входят все процедуры и функции стандартного Паскаля, а также встроенные процедуры и функции, которые не вошли в другие стандартные модули (например, INC, DEC, GETDIR и т.п.). Как уже отмечалось, модуль SYSTEM подключается к любой программе независимо от того, объявлен ли он в предложении USES или нет, поэтому его глобальные константы, переменные и подпрограммы считаются встроенными в Турбо Паскаль.

Модуль PRINTER. Делает доступным вывод текстов на матричный принтер. В нем определяется файловая переменная LST типа TEXT, которая связывается с логическим устройством PRN. После подключения модуля может быть выполнена, например, такая программа:

```
Uses Printer;
begin
writeln (LST, 'Турбо Паскаль')
end.
```

Модуль CRT. В нем сосредоточены процедуры и функции, обеспечивающие управление текстовым режимом работы экрана. С помощью входящих в модуль подпрограмм можно перемещать курсор в произвольную позицию экрана, менять цвет выводимых символов и окружающего их фона, создавать окна. Кроме того, в модуль включены также процедуры «слепого» чтения клавиатуры и управления звуком.

Модуль GRAPH. Содержит обширный набор типов, констант, процедур и функций для управления графическим режимом работы экрана. С помощью подпрограмм, входящих в модуль GRAPH, можно создавать разнообразные графические изображения и выводить на экран текстовые надписи стандартными или разработанными программистом шрифтами. Подпрограммы модуля GRAPH после соответствующей настройки могут поддерживать различные типы аппаратных графических средств. Настройка на имеющиеся в распоряжении программиста технические средства графики осуществляется специальными программами - драйверами,

которые не входят в файл GRAPH. TPU, но поставляются вместе с ним.

Модуль DOS. В модуле собраны процедуры и функции, открывающие доступ программам к средствам дисковой операционной системы MS- DOS.

Модуль OVERLAY. Он необходим при разработке громоздких программ с перекрытиями. Как уже говорилось, Турбо Паскаль обеспечивает создание программ, длина которых ограничивается лишь основной оперативной памятью ПК. Операционная система MS-DOS оставляет исполняемой программе около 580 Кбайт основной памяти (без учета резидентных программ и самой системы Турбо Паскаль). Память такого размера достаточна для большинства применений, тем не менее, использование программ с перекрытиями снимает это ограничение.

Два библиотечных модуля TURBO3 и GRAPH3 введены для совместимости с ранней версией 3.0 системы Турбо Паскаль.

Контрольные вопросы:

1. Назначение и структура модулей
2. Содержание и оформление интерфейсной части
3. Содержание и оформление исполняемой части
4. Содержание и оформление инициирующей части
5. Порядок и режимы компиляции модулей
6. Состав и назначение стандартных модулей

4.2. Использование библиотеки CRT

Во многих случаях стандартные для Паскаля возможности ввода/вывода данных с помощью процедур Read, ReadLn, Write, WriteLn оказываются явно недостаточными для разработки удобных в использовании диалоговых программ. Например, процедуры Read/ReadLn вводят с клавиатуры только типизированные данные, причем с обязательным эхо-повтором набираемых символов на экране. С их помощью нельзя определить факт нажатия какой-либо специальной клавиши (функциональной клавиши, клавиши управления курсором и т.п.). Процедуры Write/WriteLn выводят сообщения, начиная с того места на экране, где в данный момент находится курсор, причем по мере вывода курсор автоматически

сдвигается на экране, а если очередной символ выводится в самом нижнем правом углу экрана, осуществляется «прокрутка» экрана: его содержимое сдвигается вверх на одну строку. Все это сильно затрудняет создание и обновление различного рода окон, меню и других атрибутов современных диалоговых программ.

Разработчики Турбо Паскаля предусмотрели несколько подпрограмм, существенно увеличивающих возможности текстового ввода/вывода [1-3, 5]. Эти подпрограммы сосредоточены в библиотеке (модуле) CRT, входящей в комплект поставки Турбо Паскаля. В модуль включены также процедуры Sound, NoSound и Delay, которые позволяют программировать звуковой генератор ПК. В этой главе обсуждается использование подпрограмм модуля CRT.

Аббревиатура CRT соответствует русскоязычной аббревиатуре ЭЛТ - электронная лучевая трубка. На профессиональном жаргоне CRT означает устройство визуализации информации (дисплей) даже в том случае, когда вместо ЭЛТ используются иные физические устройства - плазменные панели, жидкокристаллические экраны и т.п.

4.2.1. Программирование клавиатуры

Дополнительные возможности управления клавиатурой реализуются двумя функциями: KeyPressed и ReadKey.

Функция KeyPressed.

Возвращает значение типа Boolean, указывающее состояние буфера клавиатуры: False означает, что буфер пуст, а True - что в буфере есть хотя бы один символ, еще не прочитанный программой.

В MS-DOS реализуется так называемый асинхронный буферизованный ввод с клавиатуры. По мере нажатия на клавиши соответствующие коды помещаются в особый буфер, откуда они могут быть затем прочитаны программой. Стандартная длина буфера рассчитана на хранение до 16 кодов символов. Если программа достаточно долго не обращается к клавиатуре, а пользователь нажимает клавиши, буфер может оказаться переполненным. В этот момент раздается звуковой сигнал и «лишние» коды теряются. Чтение из буфера обеспечивается процедурами Read/ReadLn и функцией ReadKey. Замечу, что обращение к функции KeyPressed не задерживает исполнения программы: функция немедленно

анализирует буфер и возвращает то или иное значение, не дожидаясь нажатия клавиши.

Функция ReadKey.

Возвращает значение типа Char. При обращении к этой функции анализируется буфер клавиатуры: если в нем есть хотя бы один не прочитанный символ, код этого символа берется из буфера и возвращается в качестве значения функции, в противном случае функция будет ожидать нажатия на любую клавишу. Ввод символа с помощью этой функции не сопровождается эхо-повтором и содержимое экрана не меняется.

Пусть, например, в какой-то точке программы необходимо игнорировать все ранее нажатые клавиши, коды которых еще не прочитаны из буфера, т.е. необходимо очистить буфер. Этого можно достичь следующим способом:

```
Uses CRT;
var
C: Char;
begin
while KeyPressed do
C := ReadKey;
.....
end.
```

При использовании процедуры ReadKey необходимо учесть, что в клавиатурный буфер помещаются так называемые расширенные коды нажатых клавиш. Если нажимается любая алфавитно-цифровая клавиша, расширенный код совпадает с ASCII-кодом соответствующего символа. Например, если нажимается клавиша с латинской буквой «a» (в нижнем регистре), функция ReadKey возвращает значение chr (97), а если «A» (в верхнем регистре) - значение chr (65). При нажатии функциональных клавиш F1...F10, клавиш управления курсором, клавиш Ins, Home, Del, End, PgUp, PgDn в буфер помещается двухбайтная последовательность: сначала символ #0, а затем расширенный код клавиши. Таким образом, значение #0, возвращаемое функцией ReadKey, используется исключительно для того, чтобы указать программе на генерацию расширенного кода. Получив это значение, программа должна еще раз обратиться к функции, чтобы прочитать расширенный код клавиши.

Т.е. код сканирования клавиши. Этот код определяется порядком, в соответствии с которым микропроцессор клавиатуры Intel 8042 периодически опрашивает (сканирует) состояние клавиш.

Следующая простая программа позволит Вам определить расширенный код любой клавиши. Для завершения работы программы нажмите клавишу Esc.

```
Uses CRT;
var
C: Char;
begin
repeat
C := ReadKey;
if C<>#0 then
WriteLn(ord(C))
else
WriteLnCO1 ,ord(ReadKey) :8)
until C=#27 {27 - расширенный код клавиши Esc}
end.
```

Если Вы воспользуетесь этой программой, то обнаружите, что нажатие на некоторые клавиши игнорируется функцией ReadKey. Это прежде всего так называемые сдвиговые клавиши - Shift, Ctrl, Alt. Сдвиговые клавиши в MS-DOS обычно используются для переключения регистров клавиатуры и нажимаются в сочетании с другими клавишами. Именно таким способом, например, различается ввод прописных и строчных букв. Кроме того, функция игнорирует переключающие клавиши Caps Lock, Num. Lock, Scroll Lock, а также «лишние» функциональные клавиши F11 и F12 клавиатуры IBM AT, не имеющие аналога на клавиатуре ранних моделей IBMPC/XT (в этих машинах использовалась 84-клавишная клавиатура, в то время как на IBM AT - 101-клавишная).

В табл. 4.1 приводятся расширенные коды клавиш, возвращаемые функцией ord(ReadKey). Для режима ввода кириллицы приводятся коды, соответствующие альтернативному варианту кодировки.

Таблица 4.1

Расширенные коды клавиш

Код	Клавиша или	Код	Клавиша или

Первый байт	Второй байт	комбинация клавиш	Первый байт	Второй байт	комбинация клавиш
Алфавитно-цифровые клавиши					
8	-	Backspace (Забой)	9	-	Tab (Табуляция)
13	-	Enter	32	-	Пробел
33	-	!	34	-	"
35	-	#	36	-	$
37	-	%	38	-	&
39	-	'	40	-	(
41	-	)	42	-	*
43	-	+	44	-	,
45	-	-	46	-	.
47	-	/	48...57	-	0...9
58	-		59	-	;
60	-	<	61	-	=
62	-	>	63	-	?
64	-	@	65...90	-	A...Z
91	-	[	92	-	\
93	-	]	94	-	^
95	-		96	-	'
97...122	-	a...z	123	-	{
124	-	l	125	-	}
126	-	~	128...159	-	А...Я
160... 175	-	а...п	224...239	-	р...я
Управляющие клавиши и их сочетания со сдвиговыми					
0	3	Ctrl-2	0	15	Shift-Tab
0	16...25	Alt-Q...Alt-P (верхний ряд букв)	0	30...38	Alt-A...Alt-L (средний ряд букв)

0	44...50	Alt-Z...Alt-M (нижний ряд букв)	0	59...68	F1...F10
0	- 71	Ноте	0	72	Курсор вверх
0	73	PgUp	0	75	Курсор влево
0	77	Курсор вправо	0	79	End
0	80	Курсор вниз	0	81	PgDn
0	82	Ins	0	83	Del
0	84...93	Shift-Fl...Shift-F10	0	94...103	Ctrl-F1... Ctrl-F10
0	104...113	Alt-Fl...Alt-F10	0	114	Ctrl-PrtScr
0	115	Ctrl-курсор влево	0	116	Ctrl-Курсор вправо
0	117	Ctrl-End	0	118	Ctrl-PgDn
0	119	Ctrl-Home	0	120...131	Alt-1. ..Alt-= (верхний ряд клавиш)
0	132	Ctrl-PgUp			

4.2.2. Текстовый вывод на экран

Используемое в ПК устройство визуального отображения информации - дисплей -состоит из двух основных частей: монитора, содержащего экран (электронно-лучевую трубку или жидкокристаллическую панель) с необходимыми компонентами (устройствами развертки изображения), и блока управления, который чаще называют дисплейным адаптером или просто адаптером. Обычно оба устройства согласуются друг с другом, но в отдельных случаях этого согласования может не быть (например, цветной монитор может работать с монохромным адаптером и наоборот). Будем считать оба устройства согласованными, поэтому, говоря о различных дисплеях, я буду говорить только о различных адаптерах, так как именно в них сосредоточены основные отличия дисплеев друг от друга.

Исторически первым адаптером (1981 г.), использованным на IBM PC, был так называемый монохромный адаптер (MDA). Его возможности очень скромны: он позволял выводить только текстовые сообщения в одном из двух форматов - 25 строк по 40 или по 80 символов в строке. Символы выводились в прямом изображении (светлые символы на темном фоне), причем их ширина оставалась одинаковой в обоих режимах, поэтому при выводе в режиме 40x25 использовалась только левая половина экрана. В MDA применялись два символьных шрифта - обычный и с подчеркиванием.

В 1982 году фирма Hercules выпустила адаптер HGC (от англ. Hercules Graphics Card - графическая карта Геркулес), который полностью эмулировал MDA в текстовом режиме, но в отличие от него мог еще воспроизводить и графические изображения с разрешением 720x350 точек (пикселей).

Примерно в это же время IBM выпустила цветной графический адаптер CGA (Color Graphics Adapter) и впервые на экране ПК появился цвет. CGA позволял выводить как текстовые сообщения, так и графические изображения (с разрешением 320x200 или 640x200 пикселей). В текстовом режиме выводились 40x25 или 80x25 символов как в монохромном, так и в цветном изображениях. При использовании монохромного режима символы, в отличие от MDA, не могли подчеркиваться, зато их можно было

выводить в негативном изображении (черные символы на светлом фоне). При выводе в цветном режиме использовалось 16 цветов для символов и 8 - для окружающего их фона.

Текстовые возможности CGA стали стандартом де-факто и поддерживаются во всех последующих разработках IBM - адаптерах EGA, MCGA, VGA и SVGA. Возможности модуля CRT рассматриваются применительно к адаптерам этого типа.

Процедура TextMode.

Используется для задания одного из возможных текстовых режимов работы адаптера. Заголовок процедуры:

Procedure TextMode(Mode: Word);

Здесь Mode - код текстового режима. В качестве значения этого выражения могут использоваться следующие константы, определенные в модуле CRT:

```
const
BW40=0{Черно-белый режим 40x25}
```

Co40=1{Цветной режим 40x25}
BW80=2{Черно-белый режим 80x25}
Co80=3{Цветной режим 80x25}
Mono=7{Используется с MDA}
Font8x8=256{Используется для загружаемого шрифта в режиме 80x43
или 80x50 с адаптерами EGA илиVGA}

Код режима, установленного с помощью вызова процедуры TextMode, запоминается в глобальной переменной LastMode модуля CRT и может использоваться для восстановления начального состояния экрана.

Следующая программа иллюстрирует использование этой процедуры в различных режимах. Замечу, что при вызове TextMode сбрасываются все ранее сделанные установки цвета и окон, экран очищается и курсор переводится в его левый верхний угол.

```
Uses CRT;
Procedure Print(S: String);
(Выводит сообщение S и ждет инициативы пользователя}
begin
WriteLn(S); {Выводим сообщение}
WriteLn('Нажмите клавишу Enter...');
ReadLn {Ждем нажатия клавиши Enter}
end; {Print}
var
LM: Word; {Начальный режим экрана}
begin
LM := LastMode; {Запоминаем начальный режим работы дисплея}
TextMode(Co40);
Print('Режим 40x25");
TextMode(CoSO) ;
Print('Режим 80x25');
TextMode(Co40+Font8x8);
Print('Режим Co40+Font8x8') ;
TextMode(Co80+Font8x8);
Print('Режим Co80+Font8x8');
{Восстанавливаем исходный режим работы:}
TextMode(LM)
```

end.

Процедура TextColor.

Определяет цвет выводимых символов. Заголовок процедуры:

Procedure TextColor(Color: Byte);

Процедура TextBackground.

Определяет цвет фона. Заголовок:

Procedure TextBackground(Color: Byte);

Единственным параметром обращения к этим процедурам должно быть выражение типа Byte, задающее код нужного цвета. Этот код удобно определять с помощью следующих мнемонических констант, объявленных в модуле CRT:

```
const
Black = 0; {Черный}
Blue = 1; {Темно-синий}
Green = 2 ; {Темно-зеленый}
Cyan = 3; {Бирюзовый}
Red = 4 ; {Красный}
Magenta = 5; {Фиолетовый}
Brown = 6 ; {Коричневый}
LightGray = 7; {Светло-серый}
DarkGray = 8; {Темно-серый}
LightBlue = 9; {Синий}
LightGreen = 10; {Светло-зеленый}
LightCyan = 11; {Светло-бирюзовый}
LightRed = 12; {Розовый}
LightMagenta = 13; {Малиновый}
Yellow = 14; {Желтый}
White ' =15; {Белый}
Blink =128; {Мерцание символа}
```

Следующая программа иллюстрирует цветовые возможности Турбо Паскаля.

```
Uses CRT;
const
Col: array [1..15] of String [16] =
('темно-синий','темно-зеленый','бирюзовый','красный',
'фиолетовый','коричневый','светло-серый','темно-серый',
'синий','зеленый','светло-бирюзовый','розовый',
'малиновый','желтый','белый');
```

```
var
k: Byte;
begin
for k := 1 to 15 do
begin {Выводим 15 сообщений различными цветами}
TextColor(k);
WriteLn('Цвет ', k, ' - ',Col[k])
end;
TextColor(White+Blink); {Белые мигающие символы}
WriteLn('Мерцание символов');
{Восстанавливаем стандартный цвет}
TextColor(LightGray);
WriteLn
end.
```

Обратите внимание на последний оператор WriteLn: если его убрать, режим мерцания символов сохранится после завершения программы, несмотря на то, что перед ним стоит оператор

TextColor(LightGray)

Дело в том, что все цветовые определения предварительно заносятся в специальную переменную TextAttr модуля CRT и используются для настройки адаптера только при обращении к процедурам Write/WriteLn.

Процедура ClrScr.

Очищает экран или окно (см. ниже процедуру Window). После обращения к ней экран (окно) заполняется цветом фона и курсор устанавливается в его левый верхний угол. Например:

```
Uses CRT;
var
C: Char
begin
TextBackground(red) ;
ClrScr; {Заполняем экран красным цветом}
WriteLn('Нажмите любую клавишу...');
C := ReadKey; {Ждем нажатия любой клавиши}
TextBackground(Black) ;
ClrScr {Восстанавливаем черный фон экрана}
end.
```

Процедура Window.

Определяет текстовое окно - область экрана, которая в дальнейшем будет рассматриваться процедурами вывода как весь экран. Сразу после вызова процедуры курсор помещается в левый верхний угол окна, а само окно очищается (заполняется цветом фона). По мере вывода курсор, как обычно, смещается вправо и при достижении правой границы окна переходит на новую строку, а если он к этому моменту находился на последней строке, содержимое окна сдвигается вверх на одну строку, т.е. осуществляется «прокрутка» окна. Заголовок процедуры:

Procedure Window(XI,Y1,X2,Y2: Byte);

ЗдесьX1...Y2 - координаты левого верхнего (XI,Y1) и правого нижнего (X2,Y2) углов окна. Они задаются в координатах экрана, причем левый верхний угол экрана имеет координаты (1,1), горизонтальная координата увеличивается слева направо, а вертикальная - сверху вниз.

В следующем примере иллюстрируется вывод достаточно длинного сообщения в двух разных окнах.

```
Uses CRT;
var
k: integer;
begin
{Создаем левое окно -желтые символы на синем фоне:}
TextBackground(Blue);
Window(5,2,35,17);
TextColor(Yellow);
for k := 1 to 100 do
Write(' Нажмите клавишу Enter...');
ReadLn; {Ждем нажатия Enter}
ClrScr; {Очищаем окно}
{Создаем правое окно - белые символы на красном фоне:}
TextBackground(Red);
TextColor(White);
Window(40,2,70,17);
for k := 1 to 100 do
Write(' Нажмите клавишу Enter...');
ReadLn;
TextMode(C080) {Сбрасываем все установки}
end.
```

Обращение к процедуре Window игнорируется, если какая-либо из координат выходит за границы экрана или если нарушается одно из условий: X2>X1 и Y2>Y1. Каждое новое обращение к Window отменяет предыдущее определение окна. Границы текущего окна запоминаются в двух глобальных переменных модуля CRT: переменная WindMin типа Word хранит X1 и Y1 (XI - в младшем байте), а переменная того же типа WindMax - X2 и Y2(X2 - в младшем байте). При желании Вы можете изменять их нужным образом без обращения к Window. Например, вместо оператора

Window(40,2,70,17);

можно было бы использовать два оператора

WindMin := 39+(1 shl 8);

WindMax := 69+(16 shl 8);

(в отличие от обращения к Window координаты, хранящиеся в переменных WindMin и WindMax, соответствуют началу отсчета 0,0).

Процедура GotoXY.

Переводит курсор в нужное место экрана или текущего окна. Заголовок процедуры:

Procedure GotoXY(X,Y: Byte);

Здесь X, Y - новые координаты курсора. Координаты задаются относительно границ экрана (окна), т.е оператор

GotoXY(1,1);

означает указание перевести курсор в левый верхний угол экрана (или окна, если к этому моменту на экране определено окно). Обращение к процедуре игнорируется, если новые координаты выходят за границы экрана (окна).

Функции WhereX и WhereY.

С помощью этих функций типа Byte можно определить текущие координаты курсора: WhereX возвращает его горизонтальную, а WhereY - вертикальную координаты.

В следующей программе сначала в центре экрана создается окно, которое обводится рамкой, затем в окне выводится таблица из двух колонок.

```
Uses CRT;
const
LU =#218; {Левый верхний угол рамки}
RU =#191; {Правый верхний угол)}
LD =#192; {Левый нижний}
```

```
RD =#217; {Правый нижний}
H =#196; {Горизонтальная черта}
V =#179; {Вертикальная черта}
X1 =14; {Координаты окна}
Y1 =5;
X2 =66;
Y2 =20;
Txt = 'Нажмите клавишу Enter...';
var
k: integer;
begin
ClrScr; {Очищаем экран}
{Создаем окно в центре экрана - желтые символы на синем фоне:}
TextBackground(Blue);
TextColor(Yellow);
Window(X1,Y1,X2,У2);
ClrScr;
{Обводим окно рамкой}
Write(LU); {Левый верхний угол}
{Горизонтальная линия}
for k: = X1+1 to X2-1 do Write(H);
Write(RU); {Верхний правый угол}
for k := Y1+1 to Y2-1 do{Вертикальные линии}
begin
GotoXY(1,k-Y1+1); {Переходим к левой границе}
Write(V); {Левая черта}
GotoXY(X2-X1+1,WhereY){Правая граница}
Write(V){Правая черта}
end;
Write(LD);
{Левый нижний угол}
Window(X1,Y1,X2,Y2+1); {Расширяем вниз на одну строку координаты окна, иначе вывод в правый нижний угол вызовет прокрутку окна вверх}
GotoXY(2,Y2-Y1+1); {Возвращаем курсор из левого верхнего угла окна на нужное место}
{Горизонтальная рамка}
```

```
for k:= X1+1 to X2-1 do Write(H);
Write(RD); {Правый нижний угол}
{Определяем внутреннюю часть окна}
Window(X1+1,Y1+1,X2-1,Y2-1);
{Выводим левый столбец}
for k := Y1+1 to Y2-2 do
WriteLn('Левый столбец, строка ',k-Y1);;
{Ждем нажатия любой клавиши}
Write('Нажмите любую клавишу...');
k := ord(ReadKey); if k=0 then
k := ord(ReadKey);
DelLine; {Стираем приглашение}
{Выводим правый столбец}
for k := Y1+1 to Y2-2 do
begin
GotoXY((X2-X1) div 2,k-Y1);
Write('Правый столбец, строка ',k-Y1)
end ;
{Выводим сообщение и ждем нажатия клавиши Enter}
GotoXY((X2-X1-Length(Txt)) div 2,Y2-Y1-1);
TextColor(White);
Write(Txt);
ReadLn;
{Восстанавливаем стандартный режим}
TextMode(CO80)
end.
```

Три следующие процедуры без параметров могут оказаться полезными при разработке текстовых редакторов.

Процедура ClrEOL.

Стирает часть строки от текущего положения курсора до правой границы окна (экрана). Положение курсора не меняется.

Процедура DelLine.

Уничтожает всю строку с курсором в текущем окне (или на экране, если окно не создано). При этом все строки ниже удаляемой (если они есть) сдвигаются вверх на одну строку.

Процедура InsLine.

Вставляет строку: строка с курсором и все строки ниже ее сдвигаются вниз на одну строку; строка, вышедшая за нижнюю

границу окна (экрана), безвозвратно теряется; текущее положение курсора не меняется.

Процедуры LowVideo, NormVideo и HighVideo.

С помощью этих процедур без параметров можно устанавливать соответственно пониженную, нормальную и повышенную яркость символов. Например:

```
Uses CRT;
begin
LowVideo;
WriteLn('Пониженная яркость');
NormVideo;
WriteLn('Нормальная яркость');
HighVideo;
WriteLn('Повышенная яркость')
end.
```

Замечу, что на практике нет разницы между пониженной и нормальной яркостью изображения.

Процедура AssignCRT.

Связывает текстовую файловую переменную F с экраном с помощью непосредственного обращения к видеопамяти (т.е. к памяти, используемой адаптером для создания изображения на экране). В результате вывод в такой текстовый файл осуществляется значительно (в 3...5 раз) быстрее, чем если бы этот файл был связан с экраном стандартной процедурой Assign. Заголовок процедуры:

```
Procedure AssignCRT(F: Text);
```

В следующей программе измеряется скорость вывода на экран с помощью стандартной файловой процедуры и с помощью непосредственного обращения к видеопамяти. Вначале файловая переменная F связывается «медленной» процедурой Assign со стандартным устройством CON (т.е. с экраном) и подсчитывается количество N1 циклов вывода некоторого текста за 5*55 = 275 миллисекунд системных часов. Затем файловая переменная связывается с экраном с помощью процедуры быстрого доступа AssignCRT и точно так же подсчитывается количество N2 циклов вывода. В конце программы счетчики N1 и N2 выводятся на экран.

Замечу, что показания системных часов хранятся в оперативной памяти компьютера в виде четырехбайтного слова по адресу [$0040:$006C] и наращиваются на единицу каждые 55 миллисекунд.

```
Uses CRT;
var
F: Text;
t: LongInt; {Начало отсчета времени}
N1,N2: Word; {Счетчики вывода}
const
txt = ' Text';
begin
{----- Стандартный вывод в файл -----}
Assign(F,'CON');
Rewrite(F);
N1 := 0; {Готовим счетчик вывода}
ClrScr; {Очищаем экран}
{Запоминаем начальный момент:}
t := MemL[$0040:$006C];
{Ждем начала нового 55-мс интервала, чтобы исключить погрешность в определении времени:}
while MemL[$0040:$006C]=t do;
{Цикл вывода за 5 интервалов}
while MemL[$0040:$006C]<t+6 do
begin
inc(N1) ;
Write(F,txt)
end;
Close(F);
{----- Вывод с помощью быстрой процедуры прямого доступа к экрану - ----}
AssignCRT(F);
Rewrite(F);
N2 := 0;
ClrScr;
t := MemL[$0040:$006C];
while MemL[$0040:$006C]=t do;
while MemL[$0040:$006C]<t+6 do
begin
inc(N2);
Write(F,txt)
end ;
```

```
Close(F);
{Печатаем результат}
ClrScr;
WriteLn(Nl,N2:10)
end.
```

Следует учесть, что вывод на экран обычным образом - без использования файловой переменной (например, оператором Write (txt)) также осуществляется с помощью непосредственного доступа к видеопамяти, поэтому ценность процедуры AssignCRT весьма сомнительна. Прямой доступ к видеопамяти регулируется глобальной логической переменной DirectVideo модуля CRT: если эта переменная имеет значение True, доступ разрешен, если False - доступ к экрану осуществляется с помощью относительно медленных средств операционной системы MS-DOS. По умолчанию переменная DirectVideo имеет значение True.

4.2.3. Программирование звукового генератора

Звуковые возможности ПК основаны на одноканальном управляемом звуковом генераторе, вырабатывающем электромагнитные колебания звуковой частоты. Колебания подаются на встроенный в ПК динамик и заставляют его звучать.

В модуль CRT включены три процедуры, с помощью которых Вы сможете запрограммировать произвольную последовательность звуков.

Процедура Sound.

Заставляет динамик звучать с нужной частотой. Заголовок процедуры:

```
Procedure Sound(F: Word);
```

Здесь F - выражение типа Word, определяющее частоту звука в герцах. После обращения к процедуре включается динамик и управление немедленно возвращается в основную программу, в то время как динамик будет звучать впредь до вызова процедуры NoSound.

Процедура NoSound.

Выключает динамик. Если он к этому моменту не был включен, вызов процедуры игнорируется.

Процедура Delay.

Обеспечивает задержку работы программы на заданный интервал времени. Заголовок процедуры:

Procedure Delay(T: Word);

Здесь T - выражение типа Word, определяющее интервал времени (в миллисекундах), в течение которого задерживается выполнение следующего оператора программы.

Для генерации звукового сигнала обычно используется вызов описанных процедур по схеме Sound-Delay-NoSound. Следующая программа заставит ПК воспроизвести простую музыкальную гамму. Используемый в ней массив F содержит частоты всех полутонов в первой октаве от «до» до «си». При переходе от одной октавы к соседней частоты изменяются в два раза.

```
Uses CRT;
const
F: array [1..12] of Real =
(130.8, 138.6, 146.8, 155.6, 164.8, 174.6, 185.0, 196.0, 207.7, 220.0,
233.1, 246.9); {Массив частот 1-й октавы}
Temp = 100; {Темп исполнения}
var
k,n: Integer;
begin
{Восходящая гамма}
for k := 0 to 3 do for n := 1 to 12 do
begin
Sound(Round(F[n]*(1 shl k) )) ;
Delay(Temp);
NoSound
end ;
{Нисходящая гамма}
for k := 3 downto 0 do
for n := 12 downto 1 do
begin
Sound(Round(F[n]*(1 shl k)) ) ;
Delay(Temp);
NoSound
end
end.
```

Контрольные вопросы:

1. Назначение библиотеки CRT
2. Назначение и использование функции KeyPressed.
3. Назначение и использование функции ReadKey.
4. Назначение и использование процедуры TextMode.
5. Назначение и использование процедуры TextColor.
6. Назначение и использование процедуры TextBackground.
7. Назначение и использование процедуры ClrScr.
8. Назначение и использование процедуры Window.
9. Назначение и использование процедуры GotoXY.
10. Назначение и использование процедуры Sound.
11. Назначение и использование процедуры NoSound.
12. Назначение и использование процедуры Delay.

4.3. Использование библиотеки GRAPH

Начиная с версии 4.0, в состав Турбо Паскаля включена мощная библиотека графических подпрограмм Graph, остающаяся практически неизменной во всех последующих версиях [1-3, 5]. Библиотека содержит в общей сложности более 50 процедур и функций, предоставляющих программисту самые разнообразные возможности управления графическим экраном. Для облегчения знакомства с библиотекой все входящие в нее процедуры и функции сгруппированы по функциональному принципу.

4.3.1. Переход в графический режим и возврат в текстовый

Стандартное состояние ПК после его включения, а также к моменту запуска программы из среды Турбо Паскаля соответствует работе экрана в текстовом режиме, поэтому любая программа, использующая графические средства компьютера, должна определенным образом инициировать графический режим работы дисплейного адаптера. После завершения работы программы ПК возвращается в текстовый режим.

4.3.2. Краткая характеристика графических режимов работы дисплейных адаптеров

Настройка графических процедур на работу с конкретным адаптером достигается за счет подключения нужного графического драйвера. Драйвер - это специальная программа, осуществляющая управление теми или иными техническими средствами ПК. Графический драйвер, как это не трудно догадаться, управляет дисплейным адаптером в графическом режиме. Графические драйверы разработаны фирмой Borland практически для всех типов адаптеров. Обычно они располагаются на диске в отдельном подкаталоге BGI в виде файлов с расширением BGI (от англ.: Borland Graphics Interface - графический интерфейс фирмы Borland). Например, CGA.BGI - драйвер для CG4-адаптера, EGAVGA.BGI - драйвер для адаптеров EGA и VGA и т.п.

Выпускаемые в настоящее время ПК оснащаются адаптерами, разработанными фирмой IBM, или совместимыми с ними. Если не учитывать уже упоминавшийся монохромный адаптер MDA, все они имеют возможность работы в графическом режиме. В этом режиме экран дисплея рассматривается как совокупность очень близко расположенных точек - пикселей, светимостью которых можно управлять с помощью программы.

Графические возможности конкретного адаптера определяются разрешением экрана, т.е. общим количеством пикселей, а также количеством цветов (оттенков), которыми может светиться любой из них. Кроме того, многие адаптеры могут работать с несколькими графическими страницами. Графической страницей называют область оперативной памяти, используемая для создания «карты» экрана, т.е. содержащая информацию о светимости (цвете) каждого пикселя. Ниже приводится краткая характеристика графических режимов работы наиболее распространенных адаптеров.

Адаптер CGA (Color Graphics Adapter - цветной графический адаптер) имеет 5 графических режимов. Четыре режима соответствуют низкой разрешающей способности экрана (320 пикселей по горизонтали и 200 по вертикали, т.е. 320x200) и отличаются только набором допустимых цветов - палитрой. Каждая палитра состоит из трех цветов, а с учетом черного цвета несветящегося пикселя - из четырех: палитра 0 (светло-зеленый, розовый, желтый), палитра 1 (светло-бирюзовый, малиновый, белый),

палитра 2 (зеленый, красный, коричневый) и палитра 3 (бирюзовый, фиолетовый, светло-серый). Пятый режим соответствует высокому разрешению 640x200, но каждый пиксель в этом случае может светиться либо каким-то одним заранее выбранным и одинаковым для всех пикселей цветом, либо не светиться вовсе, т.е. палитра этого режима содержит два цвета. В графическом режиме адаптер CGA использует только одну страницу.

Адаптер EGA (Enhanced Graphics Adapter - усиленный графический адаптер) может полностью эмулировать графические режимы адаптера CGA. Кроме того, в нем возможны режимы: низкого разрешения (640x200, 16 цветов, 4 страницы) и высокого разрешения (640x350, 16 цветов, 1 страница). В некоторых модификациях используется также монохромный режим (640x350, 1 страница, 2 цвета).

Адаптер MCGA (Multi-Color Graphics Adapter - многоцветный графический адаптер) совместим с CGA и имеет еще один режим - 640x480, 2 цвета, 1 страница. Такими адаптерами оснащались младшие модели серии ПК PS/2 фирмы IBM. Старшие модели этой серии оснащаются более совершенными адаптерами VGA (Video Graphics Array -графический видеомассив. Адаптер VGA эмулирует режимы адаптеров CGA и EGA и дополняет их режимом высокого разрешения (640x480, 16 цветов, 1 страница).

Не так давно появились так называемые супер-VGA адаптеры (SVGA) с разрешением 800x600 и более, использующие 256 и более цветовых оттенков. В настоящее время эти адаптеры получили повсеместное распространение, однако в библиотеке Graph для них нет драйверов. Поскольку SVGA совместимы с VGA, для управления современными графическими адаптерами приходится использовать драйвер EGAVGA.BGI и довольствоваться его относительно скромными возможностями.

Несколько особняком стоят достаточно популярные адаптеры фирмы Hercules. Адаптер HGC имеет разрешение 720x348, его пиксели могут светиться одним цветом (обычно светло-коричневым) или не светиться вовсе, т.е. это монохромный адаптер. Адаптер HGC+ отличается несущественными усовершенствованиями, а адаптер HICC (Hercules In Color Card) представляет собой 16-цветный вариант HGC+.

4.3.3. Процедуры и функции

Процедура InitGraph.

Инициирует графический режим работы адаптера. Заголовок процедуры:

Procedure InitGraph(var Driver,Mode: Integer; Path: String);

Здесь Driver - переменная типа Integer, определяет тип графического драйвера; Mode - переменная того же типа, задающая режим работы графического адаптера; Path - выражение типа String, содержащее имя файла драйвера и, возможно, маршрут его поиска.

К моменту вызова процедуры на одном из дисковых носителей информации должен находиться файл, содержащий нужный графический драйвер. Процедура загружает этот драйвер в оперативную память и переводит адаптер в графический режим работы. Тип драйвера должен соответствовать типу графического адаптера. Для указания типа драйвера в модуле предопределены следующие константы:

```
const
Detect=0; {Режим автоопределения типа}
CGA=1;
MCGA=2;
EGA=3;
EGA64=4;
EGAMono=5;
IBM8514=6;
HercMono=7;
ATT400=8;
VGA=9;
PC3270=10;
```

Большинство адаптеров могут работать в различных режимах. Для того, чтобы указать адаптеру требуемый режим работы, используется переменная Mode, значением которой в момент обращения к процедуре могут быть такие константы:

```
const
{ Адаптер CGA : }
CGACO = 0;      {Низкое разрешение, палитра  0}
CGAC1 = 1;      {Низкое разрешение, палитра  1}
CGAC2 = 2;      {Низкое разрешение, палитра  2}
CGAC3 = 3;      {Низкое разрешение, палитра  3}
```

```
CGAHi = 4;       {Высокое разрешение}
{Адаптер MCGA:}
MCGACO = 0;  {Эмуляция CGACO}
MCGAC1 = 1;   {Эмуляция CGAC1}
MCGAC2 = 2;   {Эмуляция CGAC2}
MCGAC3 = 3;   {Эмуляция CGAC3}
MCGAMed = 4;
{Эмуляция CGAHi}
MCGAHi = 5;    {640x480}
{Адаптер EGA :}
EGALo = 0;       {640x200, 16 цветов}
EGAHi = 1;       {640x350, 16 цветов}
EGAMonoHi = 3; {640x350, 2 цвета}
{Адаптеры HGC и HGC+:}
HercMonoHi = 0; {720x348}
{АдаптерATT400:}
ATT400CO = 0; {Аналог режима CGACO}
ATT400C1 = 1; (Аналог режима CGAC1}
ATT400C2 = 2; {Аналог режима CGAC2}
ATT400C3 = 3; {Аналог режима CGAC3}
ATT400Med = 4; {Аналог режима CGAHi}
ATT400H1 = 5; {640x400, 2 цвета}
{Адаптер VGA:}
VGALo = 0; {640x200}
VGAMed = 1; {640x350}
VGAHi = 2; {640x480}
PC3270H1 = 0; {Аналог HercMonoHi}
{Адаптер 1BM8514}
IBM8514LO =0; {640x480, 256 цветов}
IBM8514H1 = 1; {1024x768, 256 цветов}
```

Пусть, например, драйвер CGA.BGI находится в каталоге TP\BGI на диске C и устанавливается режим работы 320x200 с палитрой 2. Тогда обращение к процедуре будет таким:

```
Uses Graph;
var
Driver, Mode : Integer;
begin
Driver := CGA; {Драйвер}
```

```
Mode := CGAC2; {Режим работы}
InitGraph(Driver, Mode,' C:\TP\BGI') ;
.....
```

Если тип адаптера ПК неизвестен или если программа рассчитана на работу с любым адаптером, используется обращение к процедуре с требованием автоматического определения типа драйвера:

```
Driver := Detect;
InitGraph(Driver, Mode, 'C:\TP\BGI');
```

После такого обращения устанавливается графический режим работы экрана, а при выходе из процедуры переменные Driver и Mode содержат целочисленные значения, определяющие тип драйвера и режим его работы. При этом для адаптеров, способных работать в нескольких режимах, выбирается старший режим, т.е. тот, что закодирован максимальной цифрой. Так, при работе с CGA - адаптером обращение к процедуре со значением Driver = Detect вернет в переменной Driver значение 1 (CGA) и в Mode -значение 4 (CGAHi), а такое же обращение к адаптеру VGA вернет Driver = 9 (VGA) и Mode = 2 (VGAHi).

Функция GraphResult.

Возвращает значение типа Integer, в котором закодирован результат последнего обращения к графическим процедурам. Если ошибка не обнаружена, значением функции будет ноль, в противном случае - отрицательное число, имеющее следующий смысл:

```
const
grOk = 0; {Нет ошибок}
grlnitGraph =-1; {Не инициирован графический режим}
grNotDetected =-2; {Не определен тип драйвера}
grFileNotFind =-3; {Не найден графический драйвер}
grlnvalidDriver =-4; {Неправильный тип драйвера}
grNoLoadMem =- 5; {Нет памяти для размещения драйвера}
grNoScanMem = - 6; {Нет памяти для просмотра областей}
grNoFloodMem =- 7; {Нет памяти для закраски областей}
grFontNotFound = -8; {Не найден файл со шрифтом}
grNoFontMem =- 9; {Нет памяти для размещения шрифта}
grlnvalidMode =-10; {Неправильный графический режим}
grError =-11; {Общая ошибка}
grIOError =-12; {Ошибка ввода-вывода}
```

grlnvalidFont =-13; {Неправильный формат шрифта}

grInvalidFontNum=-14; {Неправильный номер шрифта}

После обращения к функции GraphResult признак ошибки сбрасывается, поэтому повторное обращение к ней вернет ноль.

Функция GraphErrorMsg.

Возвращает значение типа String, в котором по указанному коду ошибки дается соответствующее текстовое сообщение. Заголовок функции:

Function GraphErrorMsg(Code: Integer): String;

Здесь Code - код ошибки, возвращаемый функцией GraphResult.

Например, типичная последовательность операторов для инициации графического режима с автоматическим определением типа драйвера и установкой максимального разрешения имеет следующий вид:

```
var
Driver, Mode, Error:Integer;
begin
Driver := Detect; {Автоопределение драйвера}
InitGraph(Driver, Mode,' '); {Инициируем графику}
Error := GraphResult; {Получаем результат}
if Error <> grOk then{Проверяем ошибку}
begin{Ошибка в процедуре инициации}
WriteLn(GraphErrorMsg(Error)); {Выводим сообщение}
.....
end
else{Нет ошибки}
.....
```

Чаще всего причиной возникновения ошибки при обращении к процедуре InitGraph является неправильное указание местоположения файла с драйвером графического адаптера (например, файла CGA.BGI для адаптера CGA). Настройка на местоположение драйвера осуществляется заданием маршрута поиска нужного файла в имени драйвера при вызове процедуры InitGraph. Если, например, драйвер зарегистрирован в подкаталоге DRIVERS каталога PASCAL на диске D, то нужно использовать вызов:

```
InitGraph(Driver, Mode, 'd:\Pascal\Drivers');
```

Замечание. Во всех следующих примерах процедура InitGraph вызывается с параметром Driver в виде пустой строки. Такая форма обращения будет корректна только в том случае, когда нужный файл графического драйвера находится в текущем каталоге. Для упрощения повторения примеров скопируйте файл, соответствующий адаптеру Вашего ПК, в текущий каталог.

Процедура CloseGraph.

Завершает работу адаптера в графическом режиме и восстанавливает текстовый режим работы экрана. Заголовок:

Procedure CloseGraph;

Процедура RestoreCRTMode.

Служит для кратковременного возврата в текстовый режим. В отличие от процедуры CloseGraph не сбрасываются установленные параметры графического режима и не освобождается память, выделенная для размещения графического драйвера. Заголовок:

Procedure RestoreCRTMode;

Функция GetGraphMode.

Возвращает значение типа Integer, в котором содержится код установленного режима работы графического адаптера. Заголовок:

Function GetGraphMode: Integer;

Процедура SetGraphMode.

Устанавливает новый графический режим работы адаптера. Заголовок:

Procedure SetGraphMode(Mode: Integer);

Здесь Mode - код устанавливаемого режима.

Следующая программа иллюстрирует переход из графического режима в текстовый и обратно:

```
Uses Graph;
var .
Driver, Mode, Error : Integer;
begin
{Инициируем графический режим}
Driver := Detect;
InitGraph(Driver, Mode, ");
Error := GraphResult; {Запоминаем результат}
i£ Error <> grOk then {Проверяем ошибку}
WriteLn(GraphErrorMsg(Error)) {Есть ошибка}
else
```

```
begin {Нет ошибки}
WriteLn ('Это графический режим');
WriteLn ('Нажмите "Enter"...':20);
ReadLn;
{Переходим в текстовый режим}
RestoreCRTMode;
WriteLn (' А это текстовый...');
ReadLn;
{Возвращаемся в графический режим}
SetGraphMode (GetGraphMode);
WriteLn ('Опять графический режим...');
ReadLn;
CloseGraph
end
end.
```

В этом примере для вывода сообщений как в графическом, так и в текстовом режиме используется стандартная процедура WriteLn. Если Ваш ПК оснащен нерусифицированным адаптером CGA, вывод кириллицы в графическом режиме таким способом невозможен, в этом случае замените соответствующие сообщения так, чтобы использовать только латинские буквы.

Процедура DetectGraph.

Возвращает тип драйвера и режим его работы. Заголовок:

Procedure DetectGraph(var Driver,Mode: Integer);

Здесь Driver - тип драйвера; Mode - режим работы.

В отличие от функции GetGraphMode описываемая процедура возвращает в переменной Mode максимально возможный для данного адаптера номер графического режима.

Функция GetDriverName.

Возвращает значение типа String, содержащее имя загруженного графического драйвера. Заголовок:

Function GetDriverName: String;

Функция GetMaxMode.

Возвращает значение типа Integer, содержащее количество возможных режимов работы адаптера. Заголовок:

Function GetMaxMode: Integer;

Функция GetModeName.

Возвращает значение типа String, содержащее разрешение экрана и имя режима работы адаптера по его номеру. Заголовок:

Function GetModName(ModNumber: Integer): String;

Здесь ModNumber - номер режима.

Следующая программа после инициации графического режима выводит на экран строку, содержащую имя загруженного драйвера, а также все возможные режимы его работы.

```
Uses Graph;
var
a,b: Integer;
begin
a := Detect;
InitGraph(a, b, '');
WriteLn(GetDriverName);
for a := 0 to GetMaxMode do
WriteLn(GetModeName(a):10);
ReadLn;
CloseGraph
end.
```

Процедура GetModeRange.

Возвращает диапазон возможных режимов работы заданного графического адаптера. Заголовок:

Procedure GetModeRange(Drv: Integer; var Min, Max: Integer);

Здесь Drv - тип адаптера; Min - переменная типа Integer, в которой возвращается нижнее возможное значение номера режима; Max - переменная того же типа, верхнее значение номера.

Если задано неправильное значение параметра Drv, процедура вернет в обеих переменных значение -1. Перед обращением к процедуре можно не устанавливать графический режим работы экрана. Следующая программа выводит на экран названия всех адаптеров и диапазоны возможных номеров режимов их работы.

```
Uses Graph;
var
D,L,H: Integer;
const
N: array [1..11] of String [8] =
('CGA ', 'MCGA ', 'EGA ',
'EGA64 ', 'EGAMono ', ЧВМ8514 ',
```

```
'HercMono', 'ATT400 ', 'VGA ',
'PC3270 ', 'Ошибка ');
begin
WriteLn('Адаптер Мин. Макс.');
for D := 1 to 11 do
begin
GetModeRange(D, L, H);
WriteLn(N[D], L:7, H:10)
end
end.
```

4.3.4. Координаты, окна, страницы

Многие графические процдуры и функции используют указатель текущей позиции на экране, который в отличие от текстового курсора невидим. Положение этого указателя, как и вообще любая координата на графическом экране, задается относительно левого верхнего угла, который, в свою очередь, имеет координаты 0,0. Таким образом, горизонтальная координата экрана увеличивается слева направо, а вертикальная - сверху вниз.

Функции GetMaxX и GetMaxY.

Возвращают значения типа Word, содержащие максимальные координаты экрана в текущем режиме работы соответственно по горизонтали и вертикали. Например:

```
Uses Graph;
var
a,b: Integer;
begin
a := Detect; InitGraph(a, b, '');
WriteLn(GetMaxX, GetMaxY:5);
ReadLn;
CloseGraph
end.
```

Функции GetX и GetY.

Возвращают значения типа Integer, содержащие текущие координаты указателя соответственно по горизонтали и вертикали.

Координаты определяются относительно левого верхнего угла окна или, если окно не установлено, экрана.

Процедура SetViewPort.

Устанавливает прямоугольное окно на графическом экране. Заголовок:

Procedure SetViewPort(XI,Y1,X2,Y2: Integer; ClipOn: Boolean);

Здесь X1...Y2 - координаты левого верхнего (XI,Y1) и правого нижнего (X2,Y2) углов окна; CHpOn - выражение типа Boolean, определяющее «отсечку» не умещающихся в окне элементов изображения.

Координаты окна всегда задаются относительно левого верхнего угла экрана. Если параметр ClipOn имеет значение True, элементы изображения, не умещающиеся в пределах окна, отсекаются, в противном случае границы окна игнорируются. Для управления этим параметром можно использовать такие определенные в модуле константы:

```
const
ClipOn = True; {Включить отсечку}
ClipOff = False; {Не включать отсечку}
```

Следующий пример иллюстрирует действие параметра CHpOn. Программа строит два прямоугольных окна с разными значениями параметра и выводит в них несколько окружностей. Для большей наглядности окна обводятся рамками (см. рис. 4.1).

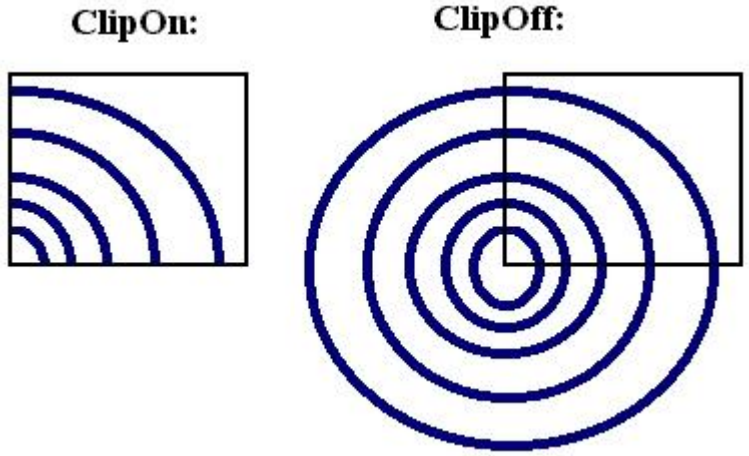


Рис. 4.1. Отсечка изображения в окне

```
Uses Graph,CRT;
var
x,y,e: Integer;
xll,yll,xl2,yl2, {Координаты 1-го окна}
x21,x22, {Левый верхний угол 2-го}
```

```
R, {Начальный радиус}
k: Integer;
begin
DirectVideo := False {Блокируем прямой доступ к видеопамяти в модуле CRT}
{Инициируем графический режим}
x := Detect; InitGraph(x, y, '');
{Проверяем результат}
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg (e) ) {Ошибка}
else
begin {Нет ошибки}
{Вычисляем координаты с учетом разрешения экрана}
x11:=GetMaxX div 60;
x12:=GetMaxX div 3;
y11:=GetMaxY div 4; y12:=2*y11;
R:=(x12-x11) div 4; x21:=x12*2;
x22:=x21+x12-x11;
{Рисуем окна}
WriteLnt'ClipOn:':10,'ClipOff:':40);
Rectangle(x11, y11, x12, y12); Rectangle(x21, y11 x22, y12);
{Назначаем 1-е окно и рисуем четыре окружности}
SetViewPort(x11, y11, x12, y12, ClipOn);
for k := 1 to 4 do
Circle(0,y11,R*k);
{Назначаем 2-е окно и рисуем окружности}
SetViewPort(x21, y11, x22, y12, ClipOff);
for k := 1 to 4 do
Circle(0,y11,R*k);
{Ждем нажатия любой клавиши}
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Процедура GetViewSettings.

Возвращает координаты и признак отсечки текущего графического окна. Заголовок:

```
Procedure GetViewSettings(var Viewlnfo: ViewPortType);
```

Здесь Viewlnfo - переменная типа ViewPortType. Этот тип в модуле Graph определен следующим образом:

```
type
ViewPortType = record
x1,y1,x2,y2: Integer; {Координаты окна}
Clip : Boolean {Признак отсечки}
end ;
```

Процедура MoveTo.

Устанавливает новое текущее положение указателя. Заголовок:

Procedure MoveTo(X,Y: integer);

Здесь X, Y - новые координаты указателя соответственно по горизонтали и вертикали.

Координаты определяются относительно левого верхнего угла окна или, если окно не установлено, экрана.

Процедура MoveRel.

Устанавливает новое положение указателя в относительных координатах.

Procedure MoveRel(DX,DY: Integer);

Здесь DX.DY- приращения новых координат указателя соответственно по горизонтали и вертикали.

Приращения задаются относительно того положения, которое занимал указатель к моменту обращения к процедуре.

Процедура ClearDevice.

Очищает графический экран. После обращения к процедуре указатель устанавливается в левый верхний угол экрана, а сам экран заполняется цветом фона, заданным процедурой SetBkColor. Заголовок:

Procedure ClearDevice;

Процедура ClearViewPort.

Очищает графическое окно, а если окно не определено к этому моменту - весь экран. При очистке окно заполняется цветом с номером О из текущей палитры. Указатель перемещается в левый верхний угол окна. Заголовок:

Procedure ClearViewPort;

В следующей программе на экране создается окно, которое затем заполняется случайными окружностями (рис. 4.2). После нажатия на любую клавишу окно очищается. Для выхода из программы нажмите Enter.

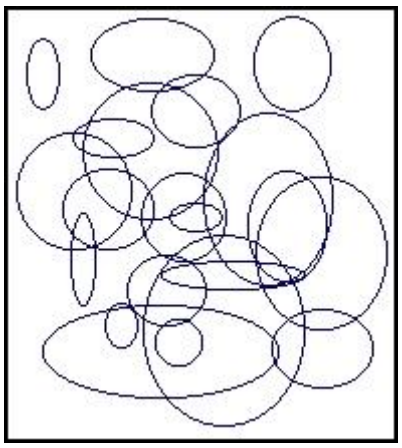


Рис. 4.2. Окно со случайными окружностями

```
Uses CRT,Graph;
var
x1,y1,x2,y2,Err: Integer;
begin
{Инициируем графический режим}
xl := Detect; InitGraph(xl,x2,'');
Err := GraphResult; if ErrogrOk then
WriteLn(GraphErrorMsg(Err))
else
begin
{Определяем координаты окна с учетом разрешения экрана}
x1 := GetMaxX div 4,-y1 := GetMaxY div 4;
x2 := 3*x1; y2 := 3*y1;
{Создаем окно}
Rectangle(x1,y1,x2,y2);
SetViewPort(x1+1,y1+1,x2-1,y2-1,ClipOn);
{Заполняем окно случайными окружностями}
repeat
Circle(Random(Ge tMaxX),Random(Ge tMaxX)
Random(GetMaxX div 5))
until KeyPressed;
{Очищаем окно и ждем нажатия Enter}
ClearViewPort;
OutTextXY(0,0,'Press Enter...1);
ReadLn;
CloseGraph
end
end.
```

Процедура GetAspectRatio.

Возвращает два числа, позволяющие оценить соотношение сторон экрана. Заголовок:

Procedure GetAspectRatio(var X,Y: Word);

Здесь X, Y - переменные типа Word. Значения, возвращаемые в этих переменных, позволяют вычислить отношение сторон графического экрана в пикселях. Найденный с их помощью коэффициент может использоваться при построении правильных геометрических фигур, таких как окружности, квадраты и т.п. Например, если Вы хотите построить квадрат со стороной L пикселей по вертикали, Вы должны использовать операторы

GetAspectRatio (Xasp, Yasp);

Rectangle(x1, y1, x1+L*round (Yasp/Xasp), y1+L);

а если L определяет длину квадрата по горизонтали, то используется оператор

Rectangle (x1,y1,x1+L,y1+L*round(Xasp/Yasp));

Процедура SetAspectRatio.

Устанавливает масштабный коэффициент отношения сторон графического экрана. Заголовок:

Procedure SetAspectRatio(X,Y: Word);

Здесь X, Y- устанавливаемые соотношения сторон.

Следующая программа строит 20 окружностей с разными соотношениями сторон экрана (рис. 4.3).

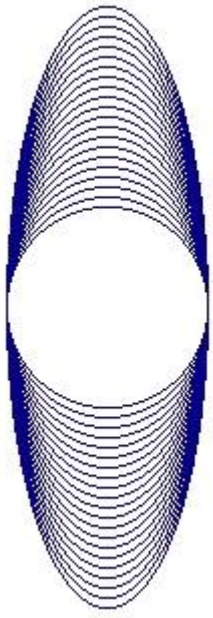


Рис. 4.3. Окружности при разных отношениях сторон экрана

```
Uses Graph,CRT;
const
```

```
R =.50;
dx = 1000;
var
d,m,e,k : Integer;
Xasp,Yasp: Word;
begin
d := Detect;
InitGraph(d, m,.");
e : = GraphResult;
if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
GetAspectRatio(Xasp, Yasp);
for k := 0 to 20 do
begin
SetAspectRatio(Xasp+k*dx,Yasp);
Circle(GetMaxX div 2,GetMaxY div 2,R)
end;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Процедура SetActivePage.

Делает активной указанную страницу видеопамяти. Заголовок:

Procedure SetActivePage(PageNum: Word);

Здесь PageNum - номер страницы.

Процедура может использоваться только с адаптерами, поддерживающими многостраничную работу (EGA, VGA и т.п.). Фактически процедура просто переадресует графический вывод в другую область видеопамяти, однако вывод текстов с помощью Write/WriteLn всегда осуществляется только на страницу, которая является видимой в данный момент (активная страница может быть невидимой). Нумерация страниц начинается с нуля.

Процедура SetVisualPage.

Делает видимой страницу с указанным номером. Обращение:

Procedure SetVisualPAge(PageNum: Word);

Здесь PageNum - номер страницы.

Процедура может использоваться только с адаптерами, поддерживающими многостраничную работу (EGA, VGA и т.п.). Нумерация страниц начинается с нуля.

Следующая программа сначала рисует квадрат в видимой странице и окружность -в невидимой. После нажатия на Enter происходит смена видимых страниц.

```
Uses Graph;
var
d,m,e: Integer;
s : String;
begin
d := Detect; InitGraph(d, m, '');
e := GraphResult; if e <> grOk then
WriteLn (GraphErrorMsg(e))
else {Нет ошибки. Проверяем, поддерживает ли драйвер многостраничную работу с видеопамятью:}
if d in [HercMono,EGA,EGA64,MCGA,VGA] then
begin {Используем многостраничный режим}
if d<>HercMono then
SetGraphMode(m-1);
{Заполняем видимую страницу}
Rectangle(10,10,GetMaxX div 2,GetMaxY div 2);
OutTextXY(0,0,'Page 0. Press Enter...');
{Заполняем невидимую}
SetActivePage (1);
Circle(GetMaxX div 2, GetMaxY div 2, 100);
OutTextXY(0,GetMaxY-10,'Page 1. Press Enter...');
{Демонстрируем страницы}
ReadLn;
SetVisualPage(1);
ReadLn;
SetVisualPage (0);
ReadLn;
CloseGraph
end
else
begin {Драйвер не поддерживает многостраничный режим}
s := GetDriverName; CloseGraph;
```

```
WriteLn('Адаптер ',s,' использует только 1 страницу')
end
end.
```

Обратите внимание на оператор

```
if doHercMono then
SetGraphMode(m-1);
```

С его помощью гарантированно устанавливается многостраничный режим работы на адаптерах EGA, MCGA, VGA. Как уже говорилось, после инициации графики с Driver=Detect устанавливается режим работы с максимально возможным номером; перечисленные адаптеры в этом режиме могут работать только с одной графической страницей, чтобы обеспечить работу с двумя страницами, следует уменьшить номер режима.

4.3.5. Линии и точки

Процедура PutPixel.

Выводит заданным цветом точку по указанным координатам. Заголовок:

```
Procedure PutPixel(X,Y: Integer; Color: Word);
```

Здесь X, Y- координаты точки; Color - цвет точки.

Координаты задаются относительно левого верхнего угла окна или, если окно не установлено, относительно левого верхнего угла экрана.

Следующая программа периодически выводит на экран «звездное небо» и затем гасит его. Для выхода из программы нажмите любую клавишу.

```
Uses CRT, Graph;
type
PixelType = record
x, y : Integer; end;
const
N = 5000; {Количество "звезд"}
var
d,r,e,k: Integer;
x1,y1,x2,y2: Integer;
a: array [1..N] of PixelType; {Координаты}
begin
```

```
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e<>grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Создаем окно в центре экрана}
x1 := GetMaxX div 4;
y1 := GetMaxY div 4;
x2 := 3*x1;
y2 := 3*y1;
Rectangle(x1,y1,x2,y2);
SetViewPort(x1+1,y1+1,x2-1,y2-1,ClipOn);
{Создаем и запоминаем координаты всех "звезд"}
for k := 1 to N do with a[k] do begin
x := Random(x2-x1);
y := Random(y2-y1)
end;
{Цикл вывода}
repeat
for k := 1 to N do
with a[k] do {Зажигаем "звезду"}
PutPixel(x,y,white);
if not KeyPressed then
for k := N downto 1 do with a[k] do {Гасим "звезду"}
PutPixel(x,y,black)
until KeyPressed;
while KeyPressed do k := ord(ReadKey);
CloseGraph
end;
end.
```

Функция GetPixel.

Возвращает значение типа Word, содержащее цвет пикселя с указанными координатами. Заголовок:

```
Function GetPixel(X,Y: Integer): Word;
```

Здесь X, Y - координаты пикселя.

Процедура Line.

Вычерчивает линию с указанными координатами начала и конца. Заголовок:

Procedure Line(X1,Y1,X2,Y2: Integer);

Здесь XL. .Yl - координаты начала (XI, Y1) и конца (X2, Y2) линии.

Линия вычерчивается текущим стилем и текущим цветом. В следующей программе в центре экрана создается окно, которое затем расчерчивается случайными линиями. Для выхода из программы нажмите любую клавишу.

```
Uses CRT, Graph;
var
d,r,e : Integer;
x1,y1,x2,y2: Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Создаем окно в центре экрана}
x1 := GetMaxX div 4;
y1 := GetMaxY div 4;
x2 := 3*x1;
y2 := 3*y1;
Rectangle(x1,y1,x2,y2);
SetViewPort(x1+1,y1+1,x2-1,y2-1,ClipOn);
{Цикл вывода случайных линий}
repeat
SetColor(succ(Random(16))); {Случайный цвет}
Line(Random(x2-x1), Random(y2-y1),
Random(x2-x1), Random(y2-y1))
until KeyPressed;
if ReadKey=#0 then d:= ord(ReadKey);
CloseGraph
end
end.
```

Процедура LineTo.

Вычерчивает линию от текущего положения указателя до положения, заданного его новыми координатами. Заголовок:

Procedure LineTo(X,Y: Integer);

Здесь X, Y - координаты нового положения указателя, они же - координаты второго конца линии.

Процедура LineRel.

Вычерчивает линию от текущего положения указателя до положения, заданного приращениями его координат. Заголовок:

Procedure LineRel (DX, DY: Integer);

Здесь DX, DY- приращения координат нового положения указателя. В процедурах LineTo и LineRel линия вычерчивается текущим стилем и текущим цветом.

Процедура SetLineStyle.

Устанавливает новый стиль вычерчиваемых линий. Заголовок:

Procedure SetLineStyle(Type,Pattern,Thick: Word)

Здесь Type, Pattern, Thick - соответственно тип, образец и толщина линии. Тип линии может быть задан с помощью одной из следующих констант:

```
const
SolidLn= 0; {Сплошная линия}
DottedLn= 1; {Точечная линия}
CenterLn= 2; {Штрих-пунктирная линия}
DashedLn= 3; {Пунктирная линия}
UserBitLn= 4; {Узор линии определяет пользователь}
```

Параметр Pattern учитывается только для линий, вид которых определяется пользователем (т.е. в случае, когда Type = UserBitLn). При этом два байта параметра Pattern определяют образец линии: каждый установленный в единицу бит этого слова соответствует светящемуся пикселю в линии, нулевой бит - несветящемуся пикселю. Таким образом, параметр Pattern задает отрезок линии длиной в 16 пикселей. Этот образец периодически повторяется по всей длине линии.

Параметр Thick может принимать одно из двух значений:

```
const
NormWidth = 1; {Толщина в один пиксель}
ThickWidth = 3; {Толщина в три пикселя}
```

Отметим, что установленный процедурой стиль линий (текущий стиль) используется при построении прямоугольников, многоугольников и других фигур.

В следующем примере демонстрируются линии всех стандартных стилей, затем вводятся слово-образец и линия с этим образцом заполнения (рис. 4.4). Для выхода из программы введите ноль.

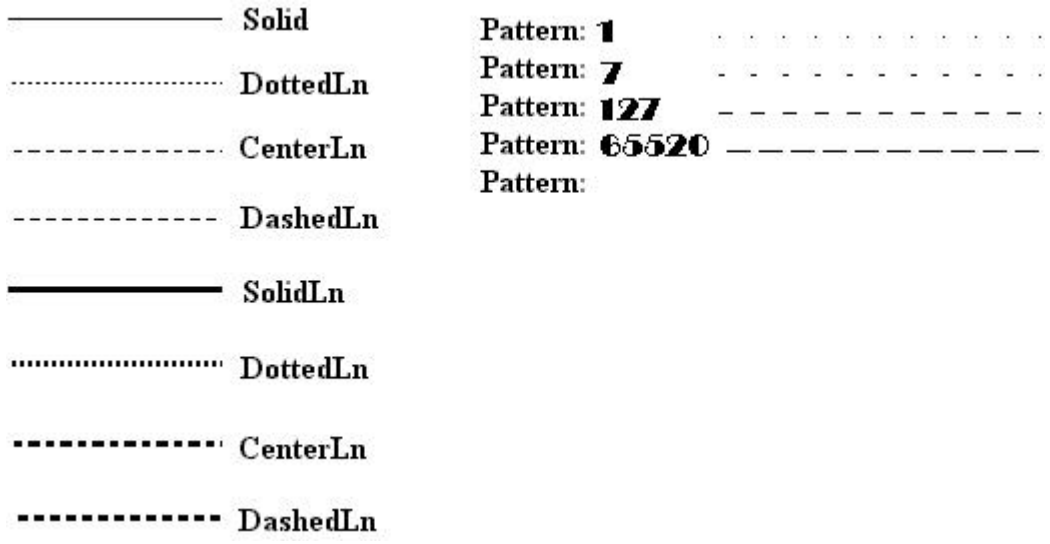


рис. 4.4. Образцы линий

```
Uses CRT, Graph;
const
style: array [0..4] of String [9] = (
'SolidLn ', 'DottedLn ', 'CenterLn 'DashedLn', 'UserBitLn');
var
d,r,e,i,j,dx,dy: Integer;
p: Word;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn (GraphErrorMsg(e))
else
begin
{Вычисляем смещение линий}
dx := GetMaxX div 6;
dy := GetMaxY div 10;
{Выводим стандартные линии}
for j := 0 to 1 do {Для двух толщин}
```

```
begin
for i := 0 to 3 do {Четыре типа линий}
begin
SetLineStyle(i, 0, j*2+1);
Line(0,(i+j*4+l)*dy,dx,(i+j*4+l)*dy);
OutTextXY(dx+10, (i+j*4+l)*dy,style [i])
end
end;
{Вводим образец и чертим линию}
 j := 0;
dy := (GetMaxY+1) div 25;
repeat
OutTextXY(320,j*dy,'Pattern: ');
GotoXY(50,j+1);
ReadLn(p); if p <> 0 then
begin
SetLineStyle(UserBitLn,p,NormWidth);
Line(440,j*dy+4, 600, j*dy+4);
inc(j)
end
until p = 0;
CloseGraph
end
end.
```

Процедура GetLineSettings.

Возвращает текущий стиль линий. Заголовок:

Procedure GetLineSettings(var Stylelnfo: LineSettingsType)

Здесь Stylelnfo - переменная типа LineSettingsType, в которой возвращается текущий стиль линий.

Тип LineSettingsType определен в модуле Graph следующим образом:

```
type
LineSettingsType = record
LineStyle: Word; {Тип линии}
Pattern : Word; {Образец}
Thickness: Word {Толщина}
end;
```

Процедура SetWriteMode.

Устанавливает способ взаимодействия вновь выводимых линий с уже существующим на экране изображением. Заголовок:

Procedure SetWriteMode(Mode);

Здесь Mode - выражение типа Integer, задающее способ взаимодействия выводимых линий с изображением.

Если параметр Mode имеет значение 0, выводимые линии накладываются на существующее изображение обычным образом (инструкцией MOV центрального процессора). Если значение 1, то это наложение осуществляется с применением логической операции XOR (исключительное ИЛИ): в точках пересечения выводимой линии с имеющимся на экране изображением светимость пикселей инвертируется на обратную, так что два следующих друг за другом вывода одной и той же линии на экран не изменят его вид.

Режим, установленный процедурой SetWriteMode, распространяется на процедуры Drawpoly, Line, LineRel, LineTo и Rectangle. Для задания параметра Mode можно использовать следующие определенные в модуле константы:

```
const
CopyPut = 0; {Наложение операцией MOV}
XORPut = 1; {Наложение операцией XOR}
```

В следующем примере на экране имитируется вид часового циферблата (рис. 4.5). Для наглядной демонстрации темп хода «часов» ускорен в 600 раз (см. оператор Delay (100)). При желании Вы сможете легко усложнить программу, связав ее показания с системными часами и добавив секундную стрелку. Для выхода из программы нажмите на любую клавишу.

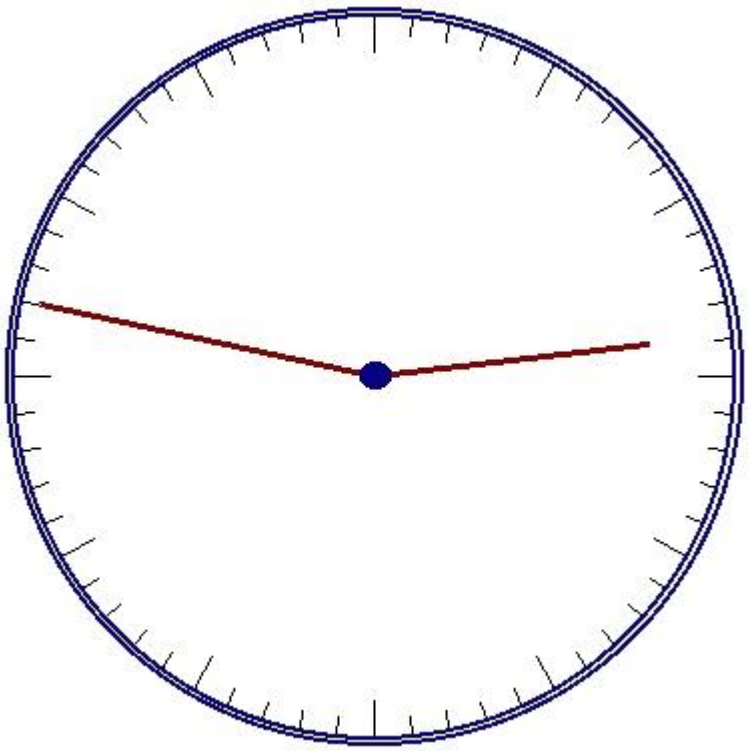


Рис. 4.5. Часовой циферблат

```
Uses Graph, CRT;
var
d,r,r1,r2,rr,k,
x1,y1,x2,y2,x01,y01: Integer;
Xasp,Yasp : Word;
begin
{Инициируем графику}
d := detect; InitGraph(d, r, '');
k := GraphResult; if k <> grOK then
WriteLn(GraphErrorMSG(k))
else
begin
{Определяем отношение сторон и размеры экрана}
x1 := GetMaxX div 2;
y1 := GetMaxY div 2;
GetAspectRatio(Xasp, Yasp);
{Вычисляем радиусы:}
r:= round(3*GetMaxY*Yasp/8/Xasp);
r1 := round(0.9*r); {Часовые деления}
r2 := round(0.95*r); {Минутные деления}
{Изображаем циферблат}
Circle(x1,y1,r); {Первая внешняя окружность}
```

```
Circle(x1,y1,round(1.02*r) ); {Вторая окружность}
for k := 0 to 59 do {Деления циферблата}
begin
if k mod 5=0 then
rr := r1 {Часовые деления}
else
rr : = r2; {Минутные деления}
{Определяем координаты концов делений}
x0l := x1+Round(rr*sin(2*pi*k/60));
y0l := y1-Round(rr*Xasp*cos(2*pi*k/60)/Yasp);
x2 := x1+Round(r*sin(2*pi*k/60));
y2 := y1-Round(r*Xasp*cos(2*pi*k/60)/Yasp);
Line(x01,y01,x2,y2) {Выводим деление}
end;
{Готовим вывод стрелок}
SetWriteMode(XORPut);
SetLineStyle(SolidLn,0,ThickWidth);
{Счетчик минут в одном часе}
{k = минуты}
r := 0;
{Цикл вывода стрелок}
repeat
for k := 0 to 59 do if not KeyPressed then begin
(Координаты часовой стрелки} x2 :=
x1+Round(0.85*r1*sin(2*pi*r/60/12));
y2 := y1-Round(0.85*r1*Xasp*cos(2*pi*r/60/12)/Yasp);
{Координаты минутной стрелки}
x01 := x1+Round(r2*sin(2*pi*k/60));
y01 := y1-Round(r2*Xasp*cos(2*pi*k/60)/Yasp);
{Изображаем стрелки}
Line(x1,y1,x2,y2);
Line(x1,y1,x01,y01) ;
Delay(100); {Для имитации реального темпа нужно установить
задержку 60000}
{Для удаления стрелок выводим их еще раз!}
Line(x1,y1,x01,y01);
Line(x1,y1,x2,y2);
{Наращиваем и корректируем счетчик минут в часе}
```

```
inc(r); if r=12*60 then
r := 0
end
until KeyPressed;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

4.3.6. Многоугольники

Процедура Rectangle.

Вычерчивает прямоугольник с указанными координатами углов. Заголовок:

```
Procedure Rectangle(X1,Y1,X2,Y2: Integer);
```

Здесь X1... Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов прямоугольника. Прямоугольник вычерчивается с использованием текущего цвета и текущего стиля линий.

В следующем примере на экране вычерчиваются 10 вложенных друг в друга прямоугольников.

```
Uses Graph, CRT;
var
d,r,e,xl,yl, x2,y2,dx,dy: Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Определяем приращения сторон}
dx := GetMaxX div 20;
dy := GetMaxY div 20;
{Чертим вложенные прямоугольники}
for d := 0 to 9 do
Rectangle(d*dx,d*dy,GetMaxX-d*dx,GetMaxY-d*dy);
if ReadKey=#0 then d := ord(ReadKey);
```

```
CloseGraph
end
end.
```

Процедура DrawPoly.

Вычерчивает произвольную ломаную линию, заданную координатами точек излома.

Procedure DrawPoly(N: Word; var Points)

Здесь N - количество точек излома, включая обе крайние точки; Points - переменная типа PointType, содержащая координаты точек излома.

Координаты точек излома задаются парой значений типа Word: первое определяет горизонтальную, второе - вертикальную координаты. Для них можно использовать следующий определенный в модуле тип:

```
type
PointType = record
x, y : Word
end;
```

При вычерчивании используется текущий цвет и текущий стиль линий. Вот как, например, можно с помощью этой процедуры вывести на экран график синуса:

```
Uses Graph;
const
N = 100; {Количество точек графика}
var
d, r, e: Integer;
m : array [O..N+1] of PointType; k : Word;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Вычисляем координаты графика}
for k := 0 to N do with m[k] do
begin
x := trunc(k*GetMaxX/N);
```

```
y := trunc(GetMaxY*(-sin(2*Pi*k/N)+1)/2)
end;
{Замыкаем график прямой линией}
m[succ(N)].x := m[0] .x;
m[succ(n)].y := m[0] .y;
DrawPoly(N + 2, m);
ReadLn;
CloseGraph
end
end.
```

В этом примере для проведения горизонтальной прямой используется «замыкание» ломаной - первая и последняя координаты ее точек излома совпадают.

Замечу, что хотя количество точек излома N - выражение типа Word, на самом деле внутри процедуры на этот параметр накладываются ограничения, связанные с конечным размером используемой буферной памяти. Вы можете убедиться в этом с помощью, например, изменения N в предыдущем примере: при N=678 график перестанет выводиться на экран, а функция GraphResult будет возвращать значение -6 (не хватает памяти для просмотра областей). Таким образом, для этой программы пороговое значение количества точек излома составляет 679. В то же время для программы

```
Uses Graph;
const
N=510; {Предельное значение, при котором на экране еще видна диагональная линия}
var
d,k: Integer;
Coo: array [1..N] of PointType;
begin
d := Detect; InitGraph(d,k,' ') ;
for k := 1 to N do with Coo[k] do
if odd(k) then
begin
X := 0;
Y := 0
end
```

```
else
begin
X := GetMaxX;
Y := GetMaxY
end;
DrawPoly(N,Coo);
ReadLn;
CloseGraph
end.
```

это значение равно 510. В этой программе ломаная задается в виде многократно накладывающихся друг на друга диагональных линий.

4.3.7. Дуги, окружности, эллипсы

Процедура Circle.

Вычерчивает окружность. Заголовок:

Procedure Circle(X,Y: Integer; R: Word);

ЗдесьX, Y- координаты центра; R - радиус в пикселях.

Окружность выводится текущим цветом. Толщина линии устанавливается текущим стилем, вид линии всегда SolidLn (сплошная). Процедура вычерчивает правильную окружность с учетом изменения линейного размера радиуса в зависимости от его направления относительно сторон графического экрана, т.е. с учетом коэффициента GetAspectRatio. В связи с этим параметр R определяет количество пикселей в горизонтальном направлении.

В следующем примере в центре экрана создается окно, постепенно заполняющееся случайными окружностями. Для выхода из программы нажмите на любую клавишу.

```
Uses Graph, CRT;
var
d,r,e,x,y: Integer;
begin.
{Инициируем графику}
d i= Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
```

```
begin
{Создаем окно в центре экрана}
x := GetMaxX div 4;
y := GetMaxY div 4;
Rectangle(x,y,3*x,3*y);
SetViewPort(x+1,y+1,3*x-1,3*y-1,ClipOn);
{Цикл вывода случайных окружностей}
repeat
SetColor(succ(Random(white))); {Случайный цвет}
SetLineStyle(0,0,2*Random(2)+1); {и стиль линии}
x := Random(GetMaxX); {Случайное положение}
y := Random(GetMaxY); {центра окружности}
Circle(x,y,Random(GetMaxY div 4));
until KeyPressed;
if ReadKey=#0 then x := ord(ReadKey);
CloseGraph
end
end.
```

Процедура Arc.

Чертит дугу окружности. Заголовок:

```
Procedure Arc(X,Y: Integer; BegA,EndA,R: Word);
```

Здесь X, Y - координаты центра; BegA, EndA - соответственно начальный и конечный углы дуги; R - радиус.

Углы отсчитываются против часовой стрелки и указываются в градусах. Нулевой угол соответствует горизонтальному направлению вектора слева направо. Если задать значения начального угла 0 и конечного - 359, то будет выведена полная окружность. При вычерчивании дуги окружности используются те же соглашения относительно линий и радиуса, что и в процедуре Circle.

Вот как выглядят две дуги: одна с углами 0 и 90, вторая 270 и 540 градусов (рис. 4.6):

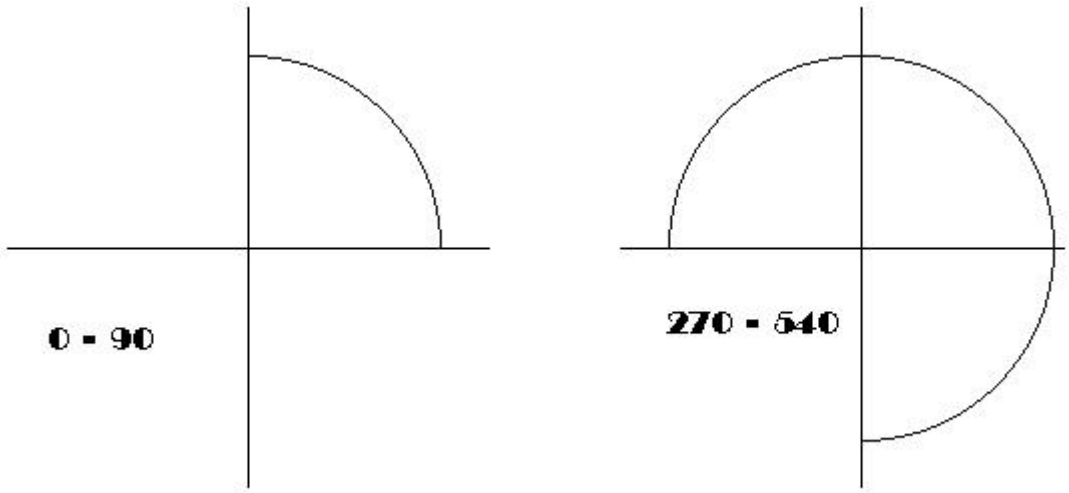


Рис. 4.6. Иллюстрация процедуры Arc

Следующая программа создает это изображение:

```
Uses Graph, CRT;
var
d, r, e : Integer;
Xasp,Yasp: Word;
begin
{Инициируем графику}
d := Detect;
InitGraphtd, r, '');
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
GetAspectRatio(Xasp,Yasp);
{R = 1/5 от вертикального размера экрана}
r := round(Yasp*GetMaxY/5/XAsp);
d := GetMaxX div 2; {Смещение второго графика}
e : = GetMaxY div 2; {Положение горизонтальной оси}
{Строим левый график}
Line (0,e,5*r div 2,e); {Горизонтальная ось}
Line (5*r div 4,e div 2,5*r div 4,3*e div 2) ;
Arc (5*r div 4,e,0,90,R); {Дуга}
OutTextXY(0,e+e div 8,'0 - 90'); {Надпись}
{Правый график}
Line (d,e,d+5*r div 2,e);
Line (d+5*r div 4,e div 2, d+5*r div 4,3*e div 2);
Arc (d+5*r div 4,e,270,540,R);
```

```
OutTextXY(d,e+e div 8,'270 - 540');
{Ждем нажатия на любую клавишу}
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Процедура GetArcCoords.

Возвращает координаты трех точек: центра, начала и конца дуги. Заголовок:

```
Procedure GetArcCoords(var Coords: ArcCoordsType);
```

Здесь Coords - переменная типа ArcCoordsType, в которой процедура возвращает координаты центра, начала и конца дуги.

Тип ArcCoordsType определен в модуле Graph следующим образом:

```
type
ArcCoordsType = record
X,Y : Integer; {Координаты центра}
Xstart,Ystart: Integer; {Начало дуги}
Xend,Yend : Integer; {Конец дуги}
end;
```

Совместное использование процедур Arc и GetArcCoords позволяет вычерчивать сопряжения двух прямых с помощью дуг. Обратите внимание на коррекцию длины радиуса в следующем примере, в котором вычерчивается прямоугольник со скругленными углами.

```
Uses Graph,CRT;
const
RadX = 50; {Горизонтальный радиус}
lx = 400; {Ширина}
ly = 100; {Высота}
var
d,r,e: Integer;
coo : ArcCoordsType;
x1,y1: Integer;
xa,ya: Word;
RadY : Integer; {Вертикальный радиус}
begin
{Инициируем графику}
```

```
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
GetAspectRatio(xa,ya) ; {Получаем отношение сторон}
{Вычисляем вертикальный радиус и положение фигуры с учетом отношения сторон экрана}
RadY := round (RadX *( xa /ya) );
x1 := (GetMaxX-lx) div 2;
y1 := (GetMaxY-2*RadY-ly) div 2;
{Вычерчиваем фигуру}
Line (x1,y1,x1+lx,y1); {Верхняя горизонтальная}
Arc (x1+lx,y1+RadY,0,90,RadX) ; {Скругление}
GetArcCoords(coo);
with coo do
begin
Line(Xstart,Ystart,Xstart,Ystart+ly);
{Правая вертикальная}
Arc(Xstart-RadX,Ystart+ly,270,0,RadX);
GetArcCoords (coo);
Line(Xstart,Ystart,Xstart-lx,Ystart);
{Нижняя горизонтальная}
Arc(Xstart-lx,Ystart-RadY,180,270,RadX);
GetArcCoords(coo);
Line(Xstart,Ystart,Xstart,Ystart-ly);
Arc(Xstart+RadX,Ystart-ly,90,180,RadX)
end ;
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Процедура Ellipse.

Вычерчивает эллипсную дугу. Заголовок:

Procedure Ellipse(X,Y: Integer; BegA,EndA,RX,RY: Word);

Здесь X, Y - координаты центра; BegA, EndA - соответственно начальный и конечный углы дуги; RX, RY- горизонтальный и вертикальный радиусы эллипса в пикселях.

При вычерчивании дуги эллипса используются те же соглашения относительно линий, что и в процедуре Circle, и те же соглашения относительно углов, что и в процедуре Arc. Если радиусы согласовать с учетом масштабного коэффициента GetAspectRatio, будет вычерчена правильная окружность.

В следующей программе вычерчиваются три эллипсных дуги (рис. 4.7) при разных отношениях радиусов. Замечу, что чем выше разрешение графического экрана, тем ближе к единице отношение сторон и тем меньше первый график отличается от третьего.

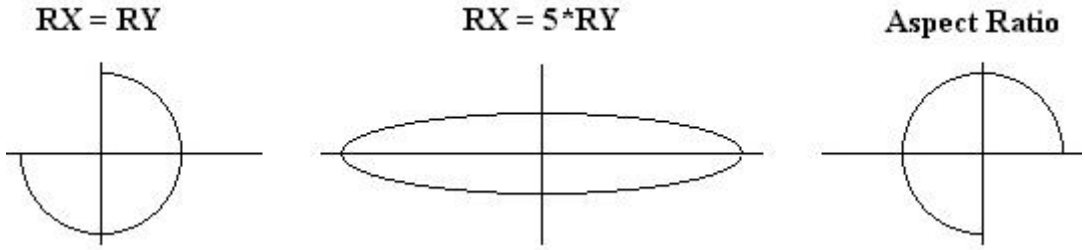


Рис. 4.7. Эллипсные дуги

```
Uses Graph, CRT;
var
d,r,e: Integer;
xa,ya: Word;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Первый график}
OutTextXY(5 0,4 0,'RX = RY'); {Надпись}
Line (0,100,160,100); {Ось X}
Line (80,55,80,145); {Ось Y}
Ellipse (80,100,180,90,40,40);
{Второй график}
OutTextXY(260,40,'RX = 5*RY');
Line (190,100,410,100);
Line (300,55,300,145);
Ellipse (300,100,0,359,100,20);
{Третий график}
OutTextXY(465,40,'Aspect Ratio');
```

```
Line (440,100,600,100);
Line (520,55,520,145);
GetAspectRatio(xa, ya);
Ellipse (520,100,0,270,40,round(40*(xa/ya)));
if ReadKey=#0 then
 d := ord(ReadKey);
CloseGraph
end
end.
```

4.3.8. Краски, палитры, заполнения

Процедура SetColor.

Устанавливает текущий цвет для выводимых линий и символов. Заголовок:

Procedure SetColor(Color: Word);

Здесь Color - текущий цвет.

В модуле Graph определены точно такие же константы для задания цвета, как и в модуле СИГ (см. п.13.2).

Функция GetColor.

Возвращает значение типа Word, содержащее код текущего цвета. Заголовок:

Function GetColor: Word;

Функция GetMaxColor.

Возвращает значение типа Word, содержащее максимальный доступный код цвета, который можно использовать для обращения к SetColor. Заголовок:

Function GetMaxColor: Word;

Процедура SetBkColor.

Устанавливает цвет фона. Заголовок:

Procedure SetBkColor(Color: Word);

Здесь Color - цвет фона.

В отличие от текстового режима, в котором цвет фона может быть только темного оттенка, в графическом режиме он может быть любым. Установка нового цвета фона немедленно изменяет цвет графического экрана. Это означает, что нельзя создать изображение, два участка которого имели бы разный цвет фона. Для CGA - адаптера в режиме высокого разрешения установка цвета фона

изменяет цвет активных пикселей. Замечу, что после замены цвета фона на любой, отличный от 0 (Black) цвет, Вы не сможете более использовать цвет 0 как черный, он будет заменяться на цвет фона, т.к. процедуры модуля Graph интерпретируют цвет с номером 0 как цвет фона. Это означает, в частности, что Вы уже не сможете вернуть фону черный цвет!

Если Ваш ПК оснащен цветным экраном, следующая программа продемонстрирует работу процедуры SetBkColor. Программа выводит десять вложенных друг в друга прямоугольников, после чего циклически меняет цвет фона. Для выхода из программы достаточно нажать на любую клавишу.

```
Uses Graph, CRT;
const
NC: array [0..15] of String [12] =
('Black','Blue','Green','Cyan','Red','Magenta',
 ' Brown','LightGray','DarkGray','LightBlue',
 'LightGreen1,'LightCyan1,'LightRed',
'LightMagenta','Yellow','White');
var
d, r, e, k, color, dx, dy: Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Выводим текст в центре экрана}
OutTextXY(200,GetMaxY div 2,'BACKGROUND COLOR');
dx := GetMaxX div 30; {Приращение длины}
dy := GetMaxY div 25; {Приращение высоты}
for k := 0 to 9 do{Выводим 10 прямоугольников}
Rectangle(k*dx,k*dy,GetMaxX-k*dx,GetMaxY-k*dy);
color := black; {Начальный цвет фона}
repeat {Цикл смены фона}
SetBkColor(color) ;
SetFillStyle(0,Color);
Bar(345,GetMaxY div 2,440,GetMaxY div 2+8);
```

```
OutTextXY(345,GetMaxY div 2,NC[color]);
delay(1000);
inc(color);
if color > White then
color := Black until KeyPressed;
if ReadKey=#0 then
k := ord(ReadKey);
CloseGraph
end
end.
```

Функция GetBkColor.

Возвращает значение типа Word, содержащее текущий цвет фона. Заголовок:

```
Function GetBkColor: Word;
```

Процедура SetPalette.

Заменяет один из цветов палитры на новый цвет. Заголовок:

```
Procedure SetPalette(N: Word; Color: Shortlnt);
```

Здесь N - номер цвета в палитре; Color - номер вновь устанавливаемого цвета.

Данная процедура может работать только с адаптерами EGA или VGA. Она не должна использоваться с IBM8514 или 256-цветным вариантом VGA - для этих адаптеров предназначена особая процедура SetRGBPalette (см. ниже). Первоначальное размещение цветов в палитрах EGA/VGA соответствует последовательности их описания константами Black,....White, т.е. цвет с индексом 0 - черный, 1 - синий, 2 - зеленый и т.д. После обращения к процедуре все фрагменты изображения, выполненные цветом с индексом N из палитры цветов, получат цвет Color. Например, если выполнить оператор

```
SetPalette(2,White);
```

то цвет с индексом 2 (первоначально это - бирюзовый цвет Cyan) будет заменен на белый. Замечу, что цвет с индексом 0 отождествляется с цветом фона и может изменяться наряду с любым другим цветом.

Следующая программа выводит на экран ряд прямых разного цвета и затем случайным образом меняет цвета палитры.

```
Uses Graph, CRT;
var
```

```
d,r,e,N,k,color: Integer;
Palette : PaletteTyper;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOK then
WriteLn(GraphErrorMsg(e))
else
begin
{Выбираем толстые сплошные линии}
SetLineStyle(SolidLn, 0, ThickWidth);
GetPalette(Palette) ; {Текущая палитра}
for Color := 0 to Palette.Size-1 do
begin
SetColor(Color);
Line(GetMaxX div 3,Color*10,2*GetMaxX div 3,Color*10)
end;
{Меняем палитру и ждем инициативы пользователя}
while not KeyPressed do
for e := 0 to Palette.Size-1 do
SetPalette(e,Random(Palette.Size));
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Процедура GetPalette.

Возвращает размер и цвета текущей палитры. Заголовок:

```
Procedure GetPalette(var Palettelnfo: PaletteType);
```

Здесь Palettelnfo - переменная типа PaletteType, возвращающая размер и цвета палитры.

В модуле Graph определена константа

```
const
MaxColors =15;
```

и тип

```
type
PaletteType = record
Size : Word; {Количество цветов в палитре}
Colors : array [0..MaxColors] of Shortlnt
```

```
{Номера входящих в палитру цветов}
end;
```

С помощью следующей программы можно вывести на экран номера всех возможных цветов из текущей палитры.

```
Uses Graph;
var
Palette: PaletteType;
d,r,e,k: Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
GetPalette(Palette); {Получаем палитру}
CloseGraph; {Возвращаемся в текстовый режим}
with Palette do {Выводим номера цветов}
for k := 0 to pred(Size) do
Write(Colors[k]:5);
end
end.
```

Процедура SetAllPalette.

Изменяет одновременно несколько цветов палитры. Заголовок процедуры:

```
Procedure SetAllPalette(var Palette);
```

Параметр Palette в заголовке процедуры описан как нетипизированный параметр. Первый байт этого параметра должен содержать длину N палитры, остальные N байты - номера вновь устанавливаемых цветов в диапазоне от -1 до MaxColors. Код -1 означает, что соответствующий цвет исходной палитры не меняется.

В следующей программе происходит одновременная смена сразу всех цветов палитры.

```
Uses Graph, CRT;
var
Palette: array [0..MaxColors] of Shortint;
d,r,e,k: Integer;
begin
```

```
{Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Выбираем толстые сплошные линии}
SetLineStyle(SolidLn, 0, ThickWidth);
{Выводим линии всеми доступными цветами}
for k := 1 to GetMaxColor do
begin
SetColor(k);
Line(GetMaxX div 3,k*10,2*GetMaxX div 3,k*10)
end;
Palette[0] := MaxColors; {Размер палитры}
repeat {Цикл смены палитры}
for k := 1 to MaxColors do
Palette[k] := Random(succ(MaxCoLors));
SetAllPalette(Palette)
until KeyPressed;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Функция GetPaletteSize.

Возвращает значение типа Integer, содержащее размер палитры (максимальное количество доступных цветов). Заголовок:

```
Function GetPaletteSize: Integer;
```

Процедура GetDefaultPalette.

Возвращает структуру палитры, устанавливаемую по умолчанию (в режиме автонастройки). Заголовок:

```
Procedure GetDefaultPalette(var Palette: PaletteType);
```

Здесь Palette - переменная типа PaletteType (см. процедуру GetPalette), в которой возвращаются размер и цвета палитры.

Процедура SetFillStyle.

Устанавливает стиль (тип и цвет) заполнения. Заголовок:

```
Procedure SetFillStyle(Fill,Color: Word);
```

Здесь Fill - тип заполнения; Color - цвет заполнения.

С помощью заполнения можно покрывать какие-либо фрагменты изображения периодически повторяющимся узором. Для указания типа заполнения используются следующие предварительно определенные константы:

```
const
EmptyFill = 0; {Заполнение фоном (узор отсутствует)}
SolidFill = 1; {Сплошное заполнение}
LineFill = 2; {Заполнение -------}
LtSlashFill = 3; {Заполнение ///////}
SlashFill = 4; {Заполнение утолщенными ///}
BkSlashFill = 5; {Заполнение утолщенными \\\}
LtBkSlashFill = 6; {Заполнение \\\\\\\}
HatchFill = 7; {Заполнение +++++++}
XHatchFill = 8; {Заполнение xxxxxxx}
InterleaveFill= 9; {Заполнение прямоугольную клеточку}
WideDotFill = 10; {Заполнение редкими точками}
CloseDotFill = 11; {Заполнение частыми точками}
UserFill = 12; {Узор определяется пользователем}
```

Программа из следующего примера продемонстрирует Вам все стандартные типы заполнения.

```
Uses Graph, CRT;
var
d,r,e,k,j,x,y: Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
x := GetMaxX div 6; {Положение графика}
y := GetMaxY div 5; {на экране}
for j := 0 to 2 do{Два ряда}
for k := 0 to 3 do{По четыре квадрата}
begin
Rectangle((k+1)*x,(j+1)*y,(k+2)*x,(j+2)*y);
SetFillStyle(k+j*4,j+1);
Bar((k+1)*x+1,(j+1)*y+1,(k+2)*x-1,(j+2)*y-1)
```

```
end;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Если параметр Fill имеет значение 12 (UserFill), то рисунок узора определяется программистом путем обращения к процедуре SetFillPattern.

Процедура SetFillPattern.

Устанавливает образец рисунка и цвет штриховки. Заголовок:

```
Procedure SetFillPattern(Pattern: FillPatternType;Color: Word);
```

Здесь Pattern - выражение типа FillPatternType; устанавливает образец рисунка для Fill - UserFill в процедуре SetFillStyle; Color - цвет заполнения.

Образец рисунка задается в виде матрицы из 8x8 пикселей и может быть представлен массивом из 8 байт следующего типа:

```
type
FillPatternType = array [1..8] of Byte;
```

Каждый разряд любого из этих байтов управляет светимостью пикселя, причем первый байт определяет 8 пикселей первой строки на экране, второй байт - 8 пикселей второй строки и т.д.

На рис. 4.8 показан пример двух образцов заполнения. На рисунке черточкой обозначается несветящийся пиксель, а прямоугольником - светящийся. Для каждых 8 пикселей приводится шестнадцатеричный код соответствующего байта.

Следующая программа заполняет этими образцами две прямоугольных области экрана.

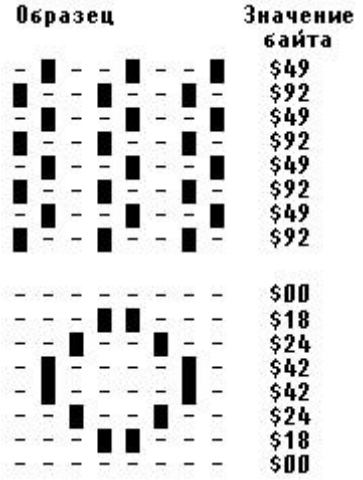


Рис. 4.8. Образцы заполнения и их коды

```
Uses Graph, CRT;
const
pattl: FillPatternType= ($49,$92,$49,$92,$49,$92,$49,$92);
patt2: FillPatternType= ($00,$18,$24,$42,$42,$24,$18,$00);
var
d,r,e: Integer;
begin  {Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
if d=CGA then
SetGraphMode (0) ; {Устанавливаем цвет для CGA}
SetFillStyle(UserFill,White);
{Левый верхний квадрат}
SetFillPattern(Patt1,1);
Bar(0,0,GetMaxX div 2, GetMaxY div 2);
{Правый нижний квадрат}
SetFillPattern(Patt2,2);
Bar(GetMaxX div 2,GetMaxY div 2,GetMaxX,GetMaxY);
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Если при обращении к процедуре указан недопустимый код цвета, вызов процедуры игнорируется и сохраняется ранее установленный образец заполнения. В частности, если в предыдущем примере убрать оператор

```
if d=CGA then
SetGraphMode(0);
```

устанавливающий цветной режим работы CGA -адаптера, на экран ПК, оснащенного адаптером этого типа, будут выведены два одинаковых прямоугольника, так как обращение

```
SetFillPattern(patt2, 2);
```

содержит недопустимо большой для данного режима код цвета и обращение игнорируется. Сказанное, однако, не относится к процедуре SetFillStyle для значения параметра Fill в диапазоне от 0 до

11: программа будет нормально работать и в режиме высокого разрешения CGA-адаптера, причем все цвета палитры, кроме цвета фона, при этом заменяются на белый.

Процедура GetFillPattern.

Возвращает образец заполнения, установленный ранее процедурой SetFillPattern. Заголовок:

Procedure GetFillPattern(var Pattern: FillPatternType);

Здесь Pattern - переменная типа FillPatternType, в которой возвращается образец заполнения.

Если программа не устанавливала образец с помощью процедуры SetFillPattern, массив Pattern заполняется байтами со значением 255 ($FF).

Процедура GetFillSettings.

Возвращает текущий стиль заполнения. Заголовок:

Procedure GetFillSettings(var Pattlnfo: FillSettingsType);

Здесь Pattlnfo - переменная типа FillSettingsType, в которой возвращается текущий стиль заполнения,

В модуле Graph определен тип:

```
type
FillSettingsType = record
Pattern: Word; {Образец}
Color : Word {Цвет}
end;
```

Поля Pattern и Color в этой, записи имеют то же назначение, что и аналогичные параметры при обращении к процедуре SetFillStyle.

Процедура SetRGBPalette.

Устанавливает цветовую гамму при работе с дисплеем IBM 8514 и адаптером VGA. Заголовок:

Procedure SetRGBPalette(ColNum,RedVal, GreenVal,BlueVal:Integer);

Здесь ColNum - номер цвета; RedVal, GreenVal, BlueVal - выражения типа Integer, устанавливающие интенсивность соответственно красной, зеленой и синей составляющих цвета.

Эта процедура может работать только с дисплеем IBM 8514, а также с адаптером VGA, использующим видеопамять объемом 256 Кбайт. В первом случае параметр ColNum задается числом в диапазоне 0...255, во втором - в диапазоне 0...15. Для установки

интенсивности используются 6 старших разрядов младшего байта любого из параметров RedVal, GreenVal, BlueVal.

В следующей программе в центре экрана выводится прямоугольник белым цветом, после чего этот цвет случайно изменяется с помощью процедуры SetRGBPalette. Для выхода из программы нужно нажать любую клавишу.

```
Uses Graph,CRT;
var
Driver, Mode, Err, xl, yl: Integer;
begin
{Инициируем графический режим}
Driver := Detect;
InitGraph(Driver, Mode, '');
Err := GraphResult;
if ErroO then
WriteLn(GraphErrorMsg(Err))
else if Driver in [IBM8514, VGA] then
begin
{Выводим прямоугольник в центре экрана}
x1 := GetMaxX div 4;
y1 := GetMaxY div 4;
SetColor(lS);
Bar(x1,y1,3*x1,3*y1);
{Изменяем белый цвет на случайный}
while not KeyPressed do
SetRGBPalette(15,Random(256),Random(256),Random(256));
CloseGraph
end
else
begin
CloseGraph; .
WriteLn('Адаптер не поддерживает ' , 'RGB-режим управления цветами')
end
end.
```

Процедура FloodFill.

Заполняет произвольную замкнутую фигуру, используя текущий стиль заполнения (узор и цвет). Заголовок:

Procedure FloodFill(X,Y: Integer; Border: Word);

Здесь X, Y- координаты любой точки внутри замкнутой фигуры; Border - цвет граничной линии.

Если фигура незамкнута, заполнение «разольется» по всему экрану.

Следует учесть, что реализованный в процедуре алгоритм просмотра границ замкнутой фигуры не отличается совершенством. В частности, если выводятся подряд две пустые строки, заполнение прекращается. Такая ситуация обычно возникает при заполнении небольших фигур с использованием типа LtSlashFill. В фирменном руководстве по Турбо Паскалю рекомендуется, по возможности, вместо процедуры FloodFill использовать FillPoly (заполнение прямоугольника).

Следующая программа демонстрирует заполнение случайных окружностей. Сначала в центре экрана создается окно, в котором заполняется небольшой прямоугольник. Часть прямоугольника останется незаполненной, в чем Вы можете убедиться, так как программа в этот момент приостанавливает работу, ожидая нажатия на клавишу Enter. Затем осуществляется вывод и заполнение случайных окружностей до тех пор, пока не будет нажата любая клавиша. Замечу, что прямоугольник заполняется полностью, если вместо типа LtSlashFill (косая штриховка линиями обычной толщины) используется SlashFill (штриховка утолщенными линиями). Если программа будет работать достаточно долго, она может «зависнуть», что лишний раз свидетельствует о несовершенстве реализованного в ней алгоритма.

```
Uses Graph, CRT;
var
d, r, e, x, y, c : Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult;
if e <> grOk then . . WriteLn(GraphErrorMsg(e))
else
begin
{Создаем прямоугольное окно}
x := GetMaxX div 4;
```

```
y. := GetMaxY div 4;
Rectangle(x,y,3*x,3*y);
SetViewPort(x+1,y+1, 3*x-1,3*y-1,ClipOn);
{Демонстрируем заливку маленького прямоугольника}
SetPillStyle(LtSlashFill,GetMaxColor);
Rectangle(0,0,8,20); FloodFill(1,1,GetMaxColor);
OutTextXY(10,25,'Press Enter...');
ReadLn; {Ждем нажатия Enter}
{Выводим окружности до тех пор, пока не будет нажата любая клавиша}
repeat
{Определяем случайный стиль заливки}
SetFillStyle(Random(12),Random(GetMaxColor+1));
{Задаем координаты центра и цвет окружности}
x := Random (GetMaxX div 2);
y := Random (GetMaxY div 2);
c := Random (succ(GetMaxColor));
SetColor(c);
{Выводим и заливаем окружность}
Circle(x, y, Random(GetMaxY div 5));
FloodFill (x, y, c)
until KeyPressed;
if ReadKey=#0 then
x := ord(ReadKey);
CloseGraph
end
end.
```

Процедура Bar.

Заполняет прямоугольную область экрана. Заголовок:

Procedure Bar(X1,Y1,X2,Y2: Integer);

Здесь XJ...Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов закрашиваемой области.

Процедура закрашивает (но не обводит) прямоугольник текущим образцом узора и текущим цветом, которые устанавливаются процедурой SetFillStyle.

Следующая программа дает красивые цветовые эффекты (закраска случайных прямоугольников).

```
Uses Graph, CRT;
```

```
var
d, r, e : Integer;
begin
{Инициируем графику}
d : = Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Создаем окно в центре экран}
d := GetMaxX div 4;
r := GetMaxY div 4; Rectangle(d,r,3*d,3*r);
SetViewPort(d+1,r+1,3*d-1,3*r-1,ClipOn);
{Цикл вывода и закраски случайных многоугольников}
repeat
SetFillStyle(Random(12),Random(succ(GetMaxColor)));
Bar(Random(Ge tMaxX),Random(Ge tMaxY),
  Random(Ge tMaxX),Random(Ge tMaxY));
until KeyPressed;
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Процедура Bar3D.

Вычерчивает трехмерное изображение параллелепипеда и закрашивает его переднюю грань . Заголовок:

Procedure Bar3D (X1,Y1,X2,Y2,Depth: Integer; Top: Boolean);

Здесь X1... Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов передней грани; Depth - третье измерение трехмерного изображения («глубина») в пикселях; Top - способ изображения верхней грани.

Если параметр Top имеет значение True, верхняя грань параллелепипеда вычерчивается, в противном случае - не вычерчивается (этот вариант используется для изображения поставленных друг на друга параллелепипедов, см. следующий пример). В качестве значения этого параметра может использоваться одна из следующих констант, определенных в модуле Graph:

```
const
```

```
TopOn = True;
TopOff = False;
```

При вычерчивании используется текущий стиль линий (SetLineStyle) и текущий цвет (SetColor). Передняя грань заливается текущим стилем заполнения (SetFillStyle).

Процедура обычно применяется при построении столбиковых диаграмм. Следует учесть, что параллелепипед «прозрачен», т.е. за его незакрашенными гранями могут быть видны другие элементы изображения.

Следующая программа иллюстрирует различные аспекты применения процедуры Bar3D.

```
Uses Graph,CRT;
var
d, r, e: Integer;
begin
{Инициируем графику}
d := Detect;
Ini-tGraph(d, r, ' ') ;
e := GraphResult;
if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Столбик с верхней гранью:}
Bar3D (80, 100, 120, 180, 15, TopOn);
{Столбик без верхней грани:}
Bar3D (150, 150, 190, 180, 15, TopOff);
{Этот столбик "стоит" на следующем и прозрачен:}
Bar3D (230, 50, 250, 150, 15, TopOn);
Bar3D (220, 150, 260, 180, 15, TopOn);
{У этого столбика нет верхней грани, и поэтому он не мешает поставленному на него сверху:}
Bar3D (300, 150, 340, 180, 15, TopOff);
SetLineStyle(3,0,1);
SetColor(Yellow);
SetFillStyle(LtSlashFill,Yellow);
Bar3D (300, 50, 340, 150, 15, TopOn);
if ReadKey=#0 then d := ord(ReadKey);
```

```
CloseGraph;
end
end.
```

Процедура FillPoly.

Обводит линией и закрашивает замкнутый многоугольник. Заголовок:

```
Procedure FillPoly(N: Word; var Coords);
```

Здесь N - количество вершин замкнутого многоугольника; Coords - переменная типа PointType, содержащая координаты вершин.

Координаты вершин задаются парой значений типа Integer: первое определяет горизонтальную, второе - вертикальную координаты. Для них можно использовать следующий определенный в модуле тип:

```
type
PointType = record
x, y : Integer
end;
```

Стиль и цвет линии контура задаются процедурами SetLineStyle и SetColor, тип и цвет заливки - процедурой SetFillStyle.

В следующем примере на экран выводятся случайные закрашенные многоугольники.

```
Uses Graph, CRT;
var
d, r, e: Integer;
p : array [1..6] of PointType; n, k : Word;
begin
{Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Создаем окно в центре экрана}
d := GetMaxX div 4;
r := GetMaxY div 4;
Rectangle(d,r,3*d,3*r);
SetViewPort(d+l,r+l,3*d-l,3*r-l,ClipOn);
```

```
{Цикл вывода случайных закрашенных многоугольников}
repeat
{Выбираем случайный цвет и узор)
SetFillStyle(Random(12),Random(succ(GetMaxColor)));
SetColor (Random(succ(GetMaxColor)));
{Назначаем случайные координаты}
n := Random (4) + 3 ; for k := 1 to n do with p[k] do
begin
x := Random (GetMaxX div 2);
y := Random (GetMaxY div 2)
end;
FillPoly (n, p) {Выводим и закрашиваем}
until KeyPressed;
if ReadKey=#0 then k := ord(ReadKey);
CloseGraph
end
end.
```

Процедура FillEllipse.

Обводит линией и заполняет эллипс. Заголовок:

```
Procedure FillEllipse(X,Y,RX,RY: Integer);
```

Здесь X, Y - координаты центра; RX, RY- горизонтальный и вертикальный радиусы эллипса в пикселях.

Эллипс обводится линией, заданной процедурами SetLineStyle и SetColor, и заполняется с использованием параметров, установленных процедурой SetFillStyle.

Процедура Sector.

Вычерчивает и заполняет эллипсный сектор. Заголовок: Procedure Sector(X,Y: Integer; BegA,EndA,RX,RY: Word);

Здесь BegA, EndA - соответственно начальный и конечный углы эллипсного сектора. Остальные параметры обращения аналогичны параметрам процедуры FillEllipse.

В следующей программе на экран выводятся случайные закрашенные эллипсы и секторы. Для выхода из программы нажмите любую клавишу.

```
Uses Graph, CRT;
var
d, r, e : Integer;
begin
```

```
{Инициируем графику}
d := Detect; InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Создаем окно в центре экрана}
d := GetMaxX div 4;
r := GetMaxY div 4;
Rectangle(d,r,3*d,3*r);
SetViewPort(d+1,r+1,3*d-1,3*r-1,ClipOn);
{Цикл вывода}
repeat
SetFillStyle(Random(12), Random(succ(GetMaxColor)));
SetColor (Random(succ(GetMaxColor)));
Sector(Random(GetMaxX div),Random(GetMaxY div 2),
Random(360),Random(360),Random(GetMaxX div 5),
Random(GetMaxY div 5));
FillEl.lipse (Random (GetMaxX div 2),
Random(GetMaxY div 2),Random(GetMaxX div 5),
Random(GetMaxY div 5))
until KeyPressed;
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Процедура PieSlice.

Вычерчивает и заполняет сектор окружности. Заголовок:

```
Procedure PieSlice(X,Y: Integer; BegA,EndA,R: Word);
```

В отличие от процедуры Sector, указывается лишь один горизонтальный радиус R, остальные параметры аналогичны параметрам процедуры Sector.

Сектор обводится линией, заданной процедурами SetLineStyle и SetColor, и заполняется с помощью параметров, определенных процедурой SetFillStyle. Процедуру удобно использовать при построении круговых диаграмм, как, например, в следующей программе (рис. 4.9).

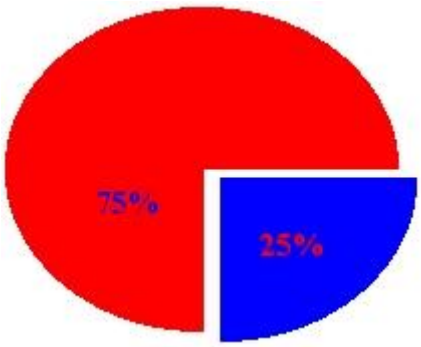


Рис. 4.9. Иллюстрация процедуры PieSlice

```
Uses Graph, CRT;
var
d, r, e : Integer;
begin
{Инициируем графический режим}
d := Detect;
InitGraph(d, r, '');
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Выводим маленький сектор}
SetFillStyle(WideDotFill, White);
PieSlice(GetMaxX div 2+5,GetMaxY div 2+4,270,360,100);
{Выводим большой сектор}
SetFillStyle (SolidFill, Red);
PieSlice (GetMaxX div 2,GetMaxY div 2, 0,270,100).;
{Выводим надписи}
OutTextXY (GetMaxX div 2+90,GetMaxY div 2+70, '25%');
OutTextXY(GetMaxX div 2-50,GetMaxY div 2-20, '75%');
{Ждем нажатия на любую клавишу}
if ReadKey=#0 then d := ord(ReadKey);
Close,Graph
end
end.
```

4.3.9. Сохранение и выдача изображений

Функция ImageSize.

Возвращает размер памяти в байтах, необходимый для размещения прямоугольного фрагмента изображения. Заголовок:

Function ImageSize(X1,Y1,X2,Y2: Integer): Word;

Здесь X1... Y2 - координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов фрагмента изображения.

Процедура GetImage.

Помещает в память копию прямоугольного фрагмента изображения. Заголовок:

Procedure GetImage(X1,Y1,X2,Y2: Integer; var Buf)

Здесь X1...Y2 - координаты углов фрагмента изображения; Buf - переменная или участок кучи, куда будет помещена копия видеопамяти с фрагментом изображения.

Размер Buf должен быть не меньше значения, возвращаемого функцией ImageSize с теми же координатами X1....Y2.

Процедура PutImage.

Выводит в заданное место экрана копию фрагмента изображения, ранее помещенную в память процедурой GetImage. Заголовок:

Procedure PutImage(X,Y: Integer; var Buf; Mode: Word);

Здесь X,Y- координаты левого верхнего угла того места на экране, куда будет скопирован фрагмент изображения; Buf - переменная или участок кучи, откуда берется изображение; Mode - способ копирования.

Как видим, координаты правого нижнего угла не указываются, так как они полностью определяются размерами вновь выводимой на экран копии изображения. Координаты левого верхнего угла могут быть какими угодно, лишь бы только выводимая копия уместилась в пределах экрана (если копия не может разместиться на экране, она не выводится и экран остается без изменений).

Параметр Mode определяет способ взаимодействия вновь размещаемой копии с уже имеющимся на экране изображением. Взаимодействие осуществляется путем применения кодируемых этим параметром логических операций к каждому биту копии и изображения. Для указания применяемой логической операции можно использовать одну из следующих предварительно определенных констант:

```
const
NormalPut= 0; {Замена существующего изображения на копию}
```

```
XorPut = 1; {Исключительное ИЛИ}
OrPut = 2; {Объединительное ИЛИ}
AndPut = 3; {Логическое И}
NotPut = 4; {Инверсия изображения}
```

Наиболее часто используются операции NormalPut, XORPut и NotPut. Первая из них просто стирает часть экрана и на это место помещает копию из памяти в том виде, как она там сохраняется. Операция NotPut делает то же самое, но копия выводится в инверсном виде. Для монохромного режима это означает замену светящихся пикселей на темные и наоборот. В цветном режиме операция NotPut применяется к коду цвета каждого пикселя. Например, для White (код 15 или в двоичном виде 1111) эта операция даст код 0000 = 0 = Black, для Red = 4 = 0100 получим 1011 = 11 = LightCyan и т.д. Операция XORPut, примененная к тому же месту экрана, откуда была получена копия, сотрет эту часть экрана. Если операцию применить дважды к одному и тому же участку, вид изображения на экране не изменится. Таким способом можно довольно просто перемещать изображения по экрану, создавая иллюзию движения.

Следующая программа рисует «Неопознанный Летающий Объект» - летающую тарелку на звездном фоне.

```
Uses Graph, CRT;
const
r = 20; {Характерный размер НЛО}
pause = 50; {Длительность паузы}
var
d,m,e,xm/ym,x,y/lx,ly,rx,ry,
Size,i,dx,dy,Width,Height: Integer;
Saucer : Pointer;
label
loop;
begin
{Инициируем графику}
d := Detect; lnitGraph(d, m, ' ') ;
e := GraphResult; if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
```

```
x := r*5;
y := r*2;
xm := GetMaxX div 4;
ym := GetMaxY div 4;
{Создаем "тарелку" из двух эллипсов с усами антенн}
Ellipse (x,y,0,360,r,r div 3+2); ,
Ellipse (x,y-4,190,357,r,r div 3);
Line (x+7,y-б,x+10,y-12);
Line (x-7,y-6, x-10, y-12);
Circle (x+10,y-12,2);
Circle (x-10,y-12,2);
FloodFill(x+l,y+4,White);
{Определяем габариты НЛО и помещаем его в кучу}
1x := x-r-1;
1y := y-14;
rx := x+r+1;
ry := y+r div 3+3;
Width := rx - lx + 1;
Height:= ry - ly + 1;
Size := ImageSize(lx, ly, rx, ry);
GetMem (Saucer, Size);
Getlmage (lx, ly, rx, ry, Saucer^);
{Стираем построенное}
Putlmage (lx, ly, Saucer^, xorPut);
{Создаем звездное небо}
Rectangle(xm,ym,3 *xm,3 *ym);
SetViewPort(xm+1,ym+1,3*xm-1,3*ym-1,ClipOn);
xm := 2*xm;
ym := 2*ym;
for i:=1 to 200 do
PutPixe1 (Random(xm), Random(ym), White) ;
{Задаем начальное положение НЛО и направление движения}
x := xm div 2;
y := ym div 2;
dx := 10;
dy := 10;
{Основной цикл}
repeat
```

```
Putlmage(x,y,Saucer^,xorPut); {Изображаем НЛО на}
Delay(pause); {новом месте и после}
Putlmage (x, y, Saucer^, XorPut); {паузы стираем его}
{Получаем новые координаты}
loop: x := x+dx;
y := y+dy;
{НЛО достиг границы экрана?}
if (x<0) or (x+Width+1>xm) or
(y<0) or (y+Height+1>ym) then
begin {Да - НЛО достиг границы: меняем направление его перемещения}
x := x-dx;
y:= y-dy;
dx : = GetMaxX div 10 - Random(GetMaxX div 5);
dy := GetMaxY div 30 - Random(GetMaxY div 15);
goto loop
end
until KeyPressed;
if ReadKey=#0 then x := ord(ReadKey);
CloseGraph
end
end.
```

4.3.10. Вывод текста

Описываемые ниже стандартные процедуры и функции поддерживают вывод текстовых сообщений в графическом режиме. Это не одно и то же, что использование процедур Write или WriteLn. Дело в том, что специально для графического режима разработаны процедуры, обеспечивающие вывод сообщений различными шрифтами в горизонтальном или вертикальном направлении, с изменением размеров и т.д. Однако в стандартных шрифтах, разработанных для этих целей фирмой Borland, отсутствует кириллица, что исключает вывод русскоязычных сообщений.

С другой стороны, процедуры Write и WriteLn после загрузки в память второй половины таблицы знакогенератора (а эта операция легко реализуется в адаптерах EGA и VGA) способны выводить

сообщения с использованием национального алфавита, но не обладают мощными возможностями специальных процедур.

Ниже описываются стандартные средства модуля Graph для вывода текста.

Процедура OutText.

Выводит текстовую строку, начиная с текущего положения указателя. Заголовок:

Procedure OutText(Txt: String);

Здесь Txt - выводимая строка.

При горизонтальном направлении вывода указатель смещается в конец выведенного текста, при вертикальном - не меняет своего положения. Строка выводится в соответствии с установленным стилем и выравниванием. Если текст выходит за границы экрана, то при использовании штриховых шрифтов он отсекается, а в случае стандартного шрифта не выводится.

Процедура OutTextXY.

Выводит строку, начиная с заданного места. Заголовок:

Procedure OutTextXY (X,Y: Integer; Txt: String);

Здесь X, Y - координаты точки вывода; Txt - выводимая строка. Отличается от процедуры OutText только координатами вывода. Указатель не меняет своего положения.

Процедура SetTextStyle.

Устанавливает стиль текстового вывода на графический экран. Заголовок:

Procedure SetTextStyle(Font,Direct,Size: Word);

Здесь Font - код (номер) шрифта; Direct - код направления; Size - код размера шрифта.

Для указания кода шрифта можно использовать следующие предварительно определенные константы:

```
const
DefaultFont = 0; {Точечный шрифт 8x8}
TriplexFont = 1; {Утроенный шрифт TRIP.CHR}
SmallFont = 2; {Уменьшенный шрифт LITT.CHR}
SansSerifFont = 3; {Прямой шрифт SANS.CHR}
GothicFont = 4; {Готический шрифт GOTH.CHR}
```

Замечу, что эти константы определяют все шрифты для версий 4.0, 5.0, 5.5 и 6.0. В версии 7,0 набор шрифтов значительно расширен, однако для новых шрифтов не предусмотрены

соответствующие мнемонические константы. В этой версии помимо перечисленных Вы можете при обращении к SetTextStyle использовать такие номера шрифтов:

Номер	Файл	Краткое описание
5	scri.chr	«Рукописный» шрифт
6	simp.chr	Одноштриховый шрифт типа Courier
7	tscr.chr	Красивый наклонный шрифт типа Times Italic
8	Icom.chr	Шрифт типа Times Roman
9	euro . chr	Шрифт типа Courier увеличенного размера
10	bold.chr	Крупный двухштриховый шрифт

Шрифт DefaultFont входит в модуль Graph и доступен в любой момент. Это -единственный матричный шрифт, т.е. его символы создаются из матриц 8x8 пикселей. Все остальные шрифты - векторные: их элементы формируются как совокупность векторов (штрихов), характеризующихся направлением и размером. Векторные шрифты отличаются более богатыми изобразительными возможностями, но главная их особенность заключается в легкости изменения размеров без существенного ухудшения качества изображения. Каждый из этих шрифтов размещается в отдельном дисковом файле. Если Вы собираетесь использовать какой-либо векторный шрифт, соответствующий файл должен находиться в Вашем каталоге, в противном случае вызов этого шрифта игнорируется и подключается стандартный.

Замечу, что шрифт DefaultFont создается графическим драйвером в момент инициации графики на основании анализа текстового шрифта. Поэтому, если Ваш ПК способен выводить кириллицу в текстовом режиме, Вы сможете с помощью этого шрифта выводить русскоязычные сообщения и в графическом режиме. В остальных шрифтах эта возможность появляется только после их модификации.

Для задания направления выдачи текста можно использовать константы:

```
const
HorizDir = 0; {Слева направо}
```

VertDir = 1; {Снизу вверх}

Как видим, стандартные процедуры OutText и OutTextXY способны выводить сообщения лишь в двух возможных направлениях - слева направо или снизу вверх. Зная структуру векторных шрифтов, нетрудно построить собственные процедуры вывода, способные выводить сообщения в любом направлении.

Каждый шрифт способен десятикратно изменять свои размеры. Размер выводимых символов кодируется параметром Size, который может иметь значение в диапазоне от 1 до 10 (точечный шрифт - в диапазоне от 1 до 32). Если значение параметра равно 0. устанавливается размер 1, если больше 10 - размер 10. Минимальный размер шрифта. при котором еще отчетливо различаются все его детали, равен 4 (для точечного шрифта - 1).

Следующая программа демонстрирует различные шрифты. Их размер выбран так. чтобы строки имели приблизительно одинаковую высоту. Перед исполнением программы скопируйте все шрифтовые файлы с расширением .CHR в текущий каталог.

```
Uses Graph, CRT;
const
FontNames: array [1..10] of String[4] =
( 'TRIP' , 'LITT" SANS ' , ' GOTH ' , 'SCRI ' , ' SIMP ' ,'TSCR ' , ' LOOM ' , ' EURO',' BOLD ' );
Tabl = 50;
Tab2 = 150;
Tab3 =220;
var
d, r, Err,{Переменные для инициации графики}
Y,dY,{Ордината вывода и ее приращение}
Size,{Размер символов}
MaxFont,{Максимальный номер шрифта}
k: Integer; {Номер шрифта}
NT, SizeT, SynibT: String; {Строки вывода}
c: Char;
 {-------------------}
Procedure OutTextWithTab ( S1, S2, S3, S4: String);
{Выводит строки S1..S4 с учетом позиций табуляции Tab1..Tab3}
begin
```

```
MoveTo( (Tab1-TextWidth(Sl) ) div2,Y);
OutText (S1) ;
MoveTo(Tabl+(Tab2-Tabl-TextWidth(S2)) div2,Y);
OutText (S2) ;
MoveTo(Tab2+(Tab3-Tab2-TextWidth(S3)) div 2,Y);
OutText(S3);
if S4='Symbols' then {Заголовок колонки Symbols}
MoveTo((Tab3+GetMaxX-TextWidth(S4)) div 2,Y)
else {Остальные строки}
MoveTo(Tab3+3,Y);
OutText(S4)
end;
{------------}
begin
{Инициируем графику}
InitGraph(d,r, ' ');
Err := GraphResult; if ErrogrOk then
WriteLn(GraphErrorMsg(Err))
else
begin
{Определяем количество шрифтов:}
{$IFDEF VER70'}
MaxFont := 10; .
{$ELSE}
MaxFont := 4;
{$ENDIF}
SetTextStyle(l,0,4);
Y := 0;
OutTextWi thTab('N','Name',Size','Symbols');
{Определяем высоту Y линии заголовка}
Y := 4*TextHeight('Z') div3;
Line(0,Y,GetMaxX,Y) ;
{Определяем начало Y таблицы и высоту dY каждой строки}
Y := 3*TextHeight('Z') div 2;
dY := (GetMaxY-Y) div (MaxFont);
{Готовим строку символов}
SymbT := '';
for c := 'a' to 'z' do
```

```
SymbT := SymbT+c;
{Цикл вывода строк таблицы}
for k := 1 to MaxFont do
begin
Size := 0;
{Увеличиваем размер до тех пор, пока высота строки не станет приблизительно равна dY}
repeat
inc(Size);
SetTextStyle(k,0,Size+1);
until (TextHeight('Z')>=dY) or (Size=10)
or (Textwidth(FontNames[k])>(Tab2-Tab1));
{Готовим номер NT и размер SizeT шрифта}
Str(k,NT);
Str(Size,SizeT);
{Выводим строку таблицы}
SetTextStyle(k,HorizDir,Size);
OutTextWithTab(NT,FontNames[k],SizeT,SymbT);
inc(Y,dY)
end;
{Рисуем линии рамки}
Rectangle(0,0,GetMaxX,GetMaxY);
Line(Tab1,0,Tabl,GetMaxY);
Line(Tab2,0,Tab2,GetMaxY);
Line(Tab3,0,TaЬ3,GetMaxY);
{Ждем инициативы пользователя}
ReadLn;
CloseGraph
end
end.
```

Процедура SetTextJustify.

Задает выравнивание выводимого текста по отношению к текущему положению указателя или к заданным координатам. Заголовок:

```
Procedure SetTextJustify(Horiz,Vert: Word);
```

Здесь Horiz - горизонтальное выравнивание; Vert - вертикальное выравнивание. Выравнивание определяет как будет размещаться

текст - левее или правее указанного места, выше, ниже или по центру. Здесь можно использовать такие константы:

```
const
LeftText = 0; {Указатель слева от текста}
CenterText= 1; {Симметрично слева и справа,верху и снизу}
RightText = 2; {Указатель справа от текста}
BottomText= 0; {Указатель снизу от текста}
TopText = 2; {Указатель сверху от текста}
```

Обратите внимание на неудачные, с моей точки зрения, имена мнемонических констант: если, например, Вы зададите LeftText, что в переводе означает «Левый Текст», сообщение будет расположено справа от текущего положения указателя (при выводе процедурой OutTextXY - справа от заданных координат). Также «наоборот» трактуются и остальные константы.

Следующая программа иллюстрирует различные способы выравнивания относительно центра графического экрана.

```
Uses Graph, CRT;
var
d, r, e : Integer;
begin
{Инициируем графику}
d := Detect; InitGraph(d,, r, ' ') ;
e := GraphResult;
if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
{Выводим перекрестие линий в центре экрана}
Line(0,GetMaxY div 2,GetMaxX,GetMaxY div 2);
Line(GetMaxX div 2,0,GetMaxX div 2,GetMaxY);
{Располагаем текст справа и сверху от центра}
SetTextStyle(TriplexFont,HorizDir,3);
SetTextJustify(LeftText,BottomText);
OutTextXY (GetMaxX div 2, GetMaxY div 2,
'LeftText,BottomText');
{Располагаем текст слева и снизу}
SetTextJustify (RightText, TopText);
```

```
OutTextXY (GetMaxX div 2, GetMaxY div 2,'RightText, TopText');
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Процедура SetUserCharSize.

Изменяет размер выводимых символов в соответствии с заданными пропорциями. Заголовок:

Procedure SetUserCharSize(XI,X2,Yl,Y2: Word);

Здесь X1...Y2 - выражения типа Word, определяющие пропорции по горизонтали и вертикали.

Процедура применяется только по отношению к векторным шрифтам. Пропорции задают масштабный коэффициент, показывающий во сколько раз увеличится ширина и высота выводимых символов по отношению к стандартно заданным значениям. Коэффициент по горизонтали находится как отношение X1 к X2, по вертикали - как отношение Y1 к Y2. Чтобы, например, удвоить ширину символов, необходимо задать X1=2 и X2=1. Стандартный размер символов устанавливается процедурой SetTextStyle, которая отменяет предшествующее ей обращение к SetUserCharSize.

В следующем примере демонстрируется изменение пропорций уменьшенного шрифта.

```
Uses Graph, CRT;
var
d, r, e : Integer;
begin
{Инициируем графику}
d := Detect; .InitGraph (d, r, '');
e := GraphResult;
if e <> grOk then
WriteLn(GraphErrorMsg(e))
else
begin
MoveTo (0, GetMaxY div 2); SetTextStyle (SmallFont, HorizDir, 5);
SetTextJustify (LeftText, BottomText);
{Выводим сообщение стандартной высотой 5}
```

```
OutText ('Normal Width,');
{Удваиваем ширину шрифта}
SetUserCharSize (2, 1, 1, 1);
OutText (' Double Width, ');
{Удваиваем высоту, возвращаем стандартную ширину}
SetUserCharSize (I, 1, 2, 1) ;
OutText ('Double Height,');
SetUserCharSize (2, 1, 2, 1) ;
OutText (' Double Width and Height');
if ReadKey=#0 then d := ord(ReadKey);
CloseGraph
end
end.
```

Функция TextWidth.

Возвращает длину в пикселях выводимой текстовой строки. Заголовок:

Function TextWidth (Txt: String): Word;

Учитываются текущий стиль вывода и коэффициенты изменения размеров символов, заданные соответственно процедурами SetTextStyle и SetUserCharSize.

Функция TextHeight.

Возвращает высоту шрифта в пикселях. Заголовок:

Function TextHeight(Txt: String): Word;

Процедура GetTextSettings.

Возвращает текущий стиль и выравнивание текста. Заголовок:

Procedure GetTextSettins(var Textlnfo: TextSettingsType);

Здесь Textlnfo - переменная типа TextSettingsType, который в модуле Graph определен следующим образом:

```
type
TextSettingsType = record
Font : Word; {Номер шрифта}
Direction: Word; {Направление}
CharSize : Word; {Код размера}
Horiz : Word; {Горизонтальное выравнивание}
Vert : Word; {Вертикальное выравнивание}
end;
```

Функция InstallUserFont.

Позволяет программе использовать нестандартный векторный шрифт. Заголовок функции:

Function InstallUserFont(FileName: String): Integer;

Здесь FileName - имя файла, содержащего векторный шрифт.

Как уже говорилось, в стандартную поставку Турбо Паскаля версий 4.0 - 6.0 включены три векторных шрифта, для версии 7.0 - 10. Функция InstallUserFont позволяет расширить этот набор. Функция возвращает идентификационный номер нестандартного шрифта, который может использоваться при обращении к процедуре SetTextStyle.

Функция InstallUserDriver.

Включает нестандартный графический драйвер в систему BGI-драйверов. Заголовок функции:

Function InstallUserDriver(FileName: String; AutoDetectPtr: Pointer): Integer;

Здесь FileName - имя файла, содержащего программу драйвера; AutoDetectPtr - адрес точки входа в специальную процедуру автоопределения типа дисплея, которая в числе прочих процедур должна входить в состав драйвера.

Эта функция расширяет и без того достаточно обширный набор стандартных графических драйверов и предназначена в основном для разработчиков аппаратных средств.

4.3.11. Включение драйвера и шрифтов в тело программы

В Турбо Паскале имеется возможность включения графического драйвера и штриховых шрифтов непосредственно в тело программы. Такое включение делает программу независимой от местоположения и наличия на диске драйверов и шрифтов, а также ускоряет подготовку графических программ к работе (шрифты и драйвер загружаются вместе с программой).

Включение драйвера и шрифтов осуществляется по следующей общей схеме. Сначала с помощью вспомогательной программы BINOBJ.EXE, входящей в комплект поставки Турбо Паскаля, драйвер и шрифты преобразуются в OBJ-файл (файл с расширением .OBJ). Для этого вне среды Турбо Паскаля необходимо вызвать утилиту BINOBJ с тремя параметрами: именем преобразуемого файла, именем

получаемого OBJ-файла и глобальным именем процедуры. Эти имена, в принципе, могут быть произвольными, правильными для MS-DOS именами. Например:

c:\tp\binobj cga.bgi cga cgadrv

В результате такого обращения из каталога TP на диске C будет вызвана программа BINOBJ и ей будут переданы следующие параметры:

CGA.BGI - имя файла с преобразуемым драйвером;

CGA - имя файла с расширением .OBJ, т.е. CGA.OBJ, который будет получен в результате исполнения программы BINOBJ;

CGADRV- глобальное имя, под которым этот драйвер будет известен программе.

После этого можно написать следующий фрагмент программы:

```
Uses Graph;
Procedure CGADRV; external;
{$L CGA.OBJ}
var
d, r, e : Integer;
begin
if RegisterBGIDriver (@CGADRV) < 0 then
begin
WriteLn ('Ошибка при регистрации драйвера');
halt
end;
d := CGA; r := CGAHi;
InitGraph (d, r, '');
.....
```

Как видно из этого примера, в программе объявляется внешняя процедура с именем CGADRV (глобальное имя, указанное при обращении к BINOBJ), причем дается директива компилятору отыскать в текущем каталоге и загрузить файл CGA.OBJ, в котором находится эта процедура. Затем осуществляется регистрация драйвера путем обращения к функции RegisterBGIDriver. Единственным параметром этой функции является адрес начала драйвера в памяти (@CGADRV). Функция возвращает значение типа Integer, которое служит для контроля правильности завершения процедуры регистрации драйвера: если это значение меньше нуля, обнаружена ошибка, в противном случае функция возвращает номер

зарегистрированного драйвера. В примере контролируется правильность регистрации драйвера и, если ошибка не обнаружена, инициируется графический режим работы экрана.

Аналогичным образом можно присоединить к программе стандартные штриховые шрифты (матричный шрифт 8x8 входит в состав модуля Graph и поэтому присоединять его не надо). Присоединение шрифта строится по описанной схеме за тем исключением, что для его регистрации вызывается функция RegisterBGIFont. Например, после преобразования

```
c:\Pascal\binobj litt.chr litt litt
```

можно использовать операторы

```
Procedure Litt;External;
{$L Litt.obj}
.....
if RegisterBGIFont (@litt) < 0 then ...
```

Обратите внимание: регистрация и драйвера, и шрифтов должна предшествовать инициации графического режима.

Регистрировать можно также драйверы (шрифты), которые не компилируются вместе с программой, а загружаются в динамическую память. Например:

```
Uses Graph;
var
p: Pointer;
f: file;
begin
Assign(f,'Litt.chr'); {Открываем файл}
Reset(f,1); {LITT.CHR для чтения}
GetMem(p,FileSize(f)) ; {Резервируем для него область кучи нужного размера}
BlockRead(f,pA,FileSize(f)){Читаем файл}
WriteLn(RegisterBGIFont (p)){Регистрируем шрифт}
end.
```

Контрольные вопросы:

1. Назначение библиотеки GRAPH
2. Назначение и использование процедуры InitGraph.

3. Назначение и использование функции GraphResult.
4. Назначение и использование функции GraphErrorMsg.
5. Назначение и использование процедуры CloseGraph.
6. Назначение и использование функций GetMaxX и GetMaxY.
7. Назначение и использование процедуры SetViewPort.
8. Назначение и использование процедуры MoveTo.
9. Назначение и использование процедуры ClearDevice.
10. Назначение и использование процедуры ClearViewPort.
11. Назначение и использование процедуры PutPixel.
12. Назначение и использование процедуры Line.
13. Назначение и использование процедуры LineTo.
14. Назначение и использование процедуры SetLineStyle.
15. Назначение и использование процедуры Rectangle.
16. Назначение и использование процедуры DrawPoly.
17. Назначение и использование процедуры Circle.
18. Назначение и использование процедуры SetColor.
19. Назначение и использование процедуры SetBkColor.
20. Назначение и использование функции GetBkColor.
21. Назначение и использование процедуры SetFillStyle.
22. Назначение и использование процедуры SetFillPattern.
23. Назначение и использование процедуры FloodFill.
24. Назначение и использование процедуры Bar.
25. Назначение и использование процедуры Bar3D.
26. Назначение и использование процедуры FillPoly.
27. Назначение и использование процедуры OutText.
28. Назначение и использование процедуры OutTextXY.
29. Назначение и использование процедуры SetTextStyle.
30. Назначение и использование функции TextWidth.
31. Назначение и использование функции TextHeight.

5. Список литературы

1. Фаронов В.В. Turbo Pascal 7.0: учебный курс. Учебное пособие для ВУЗов. КноРус, 2011.
2. Фаронов В.В. Turbo Pascal 7.0. практика программирования.(изд:7). КноРус, 2011.
3. Фаронов В.В. Turbo Pascal. Учебное пособие для ВУЗов. Питер, 2009.
4. Меняев,М.Ф. Информатика и основы программирования : учеб.пособие/ Меняев,М.Ф.. -3-е изд.,стер.. -М.: Омега-Л, 2007.
5. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. – М.: Финансы и статистика, 2006.
6. Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем: Учеб. пособие. – М.: Финансы и статистика, 2006.
7. Орлов С.А. Программная инженерия. Технологии разработки программного обеспечения – СПб.: Питер, 2016. Программирование в Delphi для Windows. Версии 2006, 2007, Turbo Delphi – М.: ЗАО «Издательство БИНОМ», 2007.
8. http://www.citforum.ru/programming/
9. http://www.cyberforum.ru/programming-theory/
10. http://pascal.sources.ru/articles/

Приложение 1.
Образец титульного листа

Министерство сельского хозяйства Российской Федерации
Департамент научно-технологической политики и образования
Красноярский государственный аграрный университет
Институт экономики и управления АПК
Кафедра Информационных технологий и математического обеспечения информационных систем

Отчет по учебной практике
по получению первичных профессиональных умений и навыков

Выполнил(а)
студент(ка) группы
_____
(Фамилия, имя, отчество)
«___»_____20__г.

Руководитель практики
(Фамилия, имя, отчество)
Оценка
_____
(Подпись)
«___»_____20__г.

Красноярск, 20_ _

Приложение 2.
ГОСТ 19.002-80

УДК 651.7/.78:002:006.354 **Группа Т55**

ГОСУДАРСТВЕННЫЙ СТАНДАРТ СОЮЗА ССР

Единая система программной документации

СХЕМЫ АЛГОРИТМОВ И ПРОГРАММ. ПРАВИЛА ВЫПОЛНЕНИЯ

ГОСТ 19.002-80
Взамен ГОСТ 19427-74

United system for program documentation.
Flowcharts. Conventions for flowcharting

Постановлением Государственного комитета СССР по стандартам от 24 апреля 1980 г. № 1867 срок введения установлен

с 01.07. 1981 г.

Настоящий стандарт распространяется на алгоритмы и программы систем программного обеспечения вычислительных машин, комплексов и систем независимо от их назначения и области применения и устанавливает правила выполнения схем алгоритмов и программ, выполняемых автоматическим способом или от руки.

Стандарт полностью соответствует МС ИСО 2636-73.

1. ПРАВИЛА ВЫПОЛНЕНИЯ СХЕМ

1.1. При выполнении схем алгоритмов и программ отдельные функции алгоритмов и программ, с учетом степени их детализации,

отображаются в виде условных графических обозначений - символов по ГОСТ 19.003-80.

Схемы должны быть выполнены на форматах по ГОСТ 2.301-68.

1.2. Для облегчения вычерчивания и нахождения на схеме символов рекомендуется поле листа разбивать на зоны. Размеры зон устанавливают с учетом минимальных размеров символов, изображенных на данном листе. Допускается один символ размещать в двух и более зонах, если размер символа превышает размер зоны.

1.3. Координаты зоны проставляют:

- по горизонтали - арабским цифрами слева направо в верхней части листа;
- по вертикали - прописными буквами латинского алфавита сверху вниз в левой части листа.

1.4. Координаты зон в виде сочетания букв и цифр присваивают символам, вписанным в поля этих зон, например А1, А2, А3, В1, В2, В3 и т. д.

При выполнении схем от руки, если поле листа не разбито на зоны, символам присваивают порядковые номера.

1.5. В пределах одной схемы, при выполнении ее от руки, допускается применять не более двух смежных размеров ряда чисел, кратных 5.

1.6. Для ускорения выполнения схем от руки рекомендуется использовать бланки с контуром прямоугольника внутри каждой зоны. Контуры не должны воспроизводиться при изготовлении копии.

1.7. Расположение символов на схеме должной соответствовать требованиям ГОСТ 19.003-80.

Исключение составляют обязательные символы «Линия потока», «Канал связи», «Комментарий» и рекомендуемые символы «Межстраничный соединитель», «Транспортирование носителей», «Материальный поток».

1.8. Линии потока должны быть параллельны линиям внешней рамки схемы.

1.9. Направления линии потока сверху вниз и слева направо принимают за основные и, если линии потока не имеют изломов, стрелками можно не обозначать. В остальных случаях направление линии потока обозначать стрелкой обязательно.

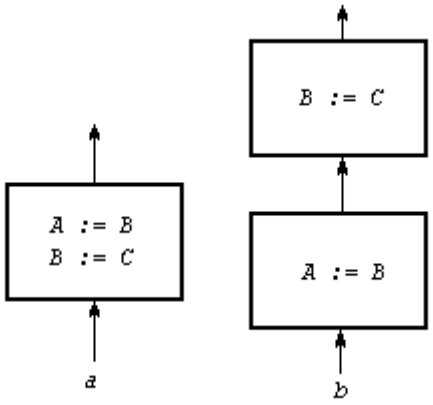
1.10. Расстояния между параллельными линиями потока должно быть не менее 3 мм, между остальными символами схемы - не менее 5 мм.

1.11. Записи внутри символа или рядом с ним должны быть выполняться машинописью с одним интервалом или чертежным шрифтом по ГОСТ 2.304-68.

1.12. Записи внутри символа или рядом с ним должны быть краткими. Сокращение слов и аббревиатуры, за исключением установленных государственными стандартами, должны быть расшифрованы в нижней части поля схемы или в документе, к которому эта схема относится.

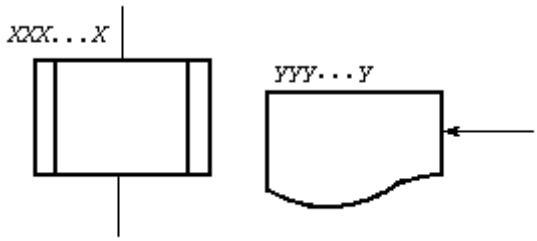
1.13. Для удобства детализации программы должны быть использованы символы «Процесс», «Решение», «Модификация», «Ввод-вывод» и «Пуск-останов», при этом внутри символа на расстоянии на менее 0,25а проводят тонкую линию (размер а по ГОСТ 19.003-80).

1.14. Записи внутри символа должны быть представлены так, чтобы их можно было читать слева направо и сверху вниз, независимо от направления потока (черт. 1). Вид *а* должен быть прочитан как вид *б*.



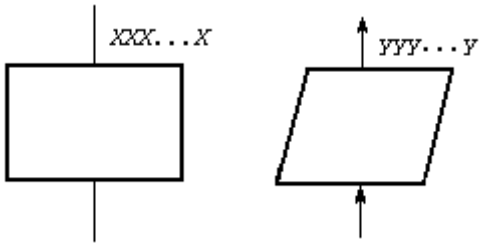
Черт. 1

1.15. В схеме символу может быть присвоен идентификатор, который должен помещаться слева над символом (например, для ссылки в других частях документации). (черт. 2).



Черт. 2

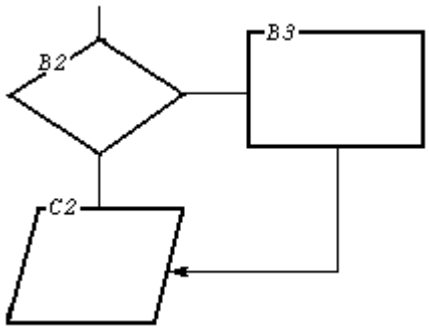
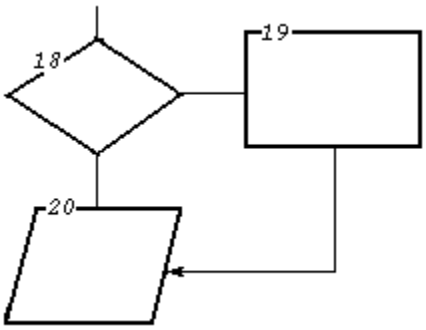
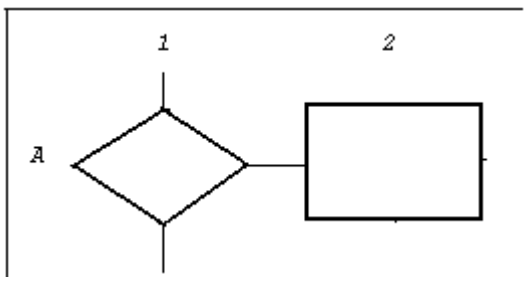
1.16. В схемах допускается краткая информация о символе (описание, уточнение или другие перекрестные ссылки для более полного понимания функции данной части системы). Описание символа должно помещаться справа над символом (черт. 3).

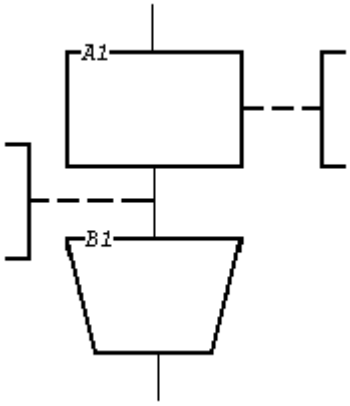
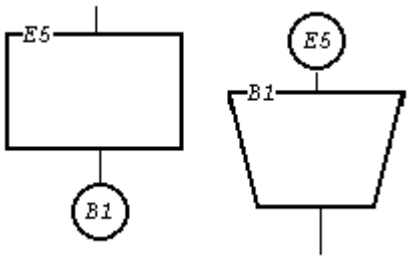
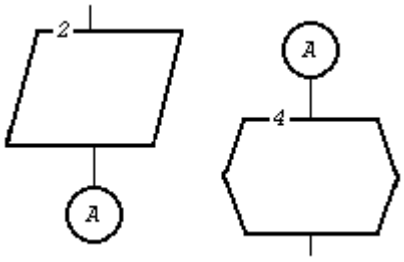


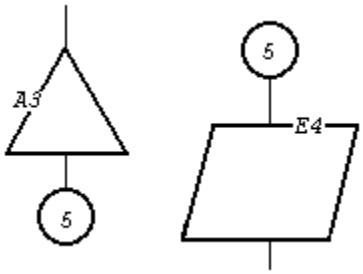
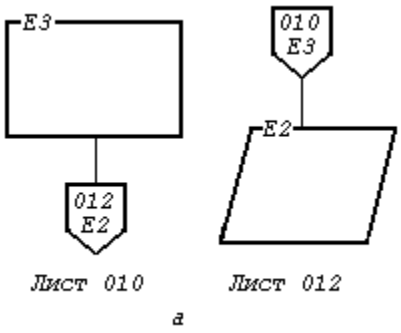
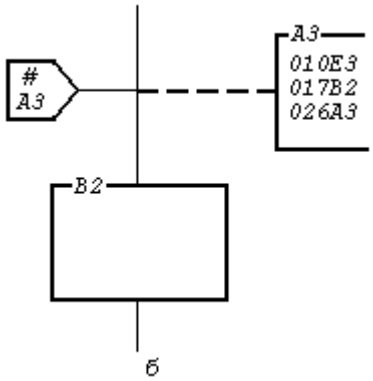
Черт. 3

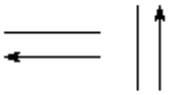
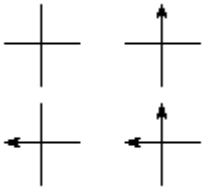
2. ПРАВИЛА ПРИМЕНЕНИЯ СИМВОЛОВ

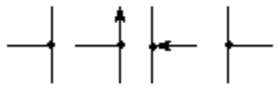
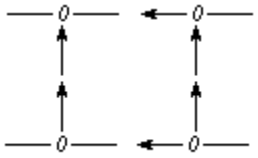
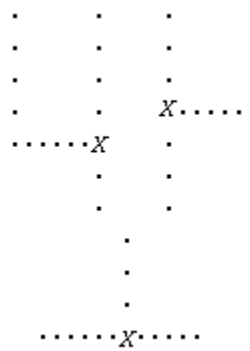
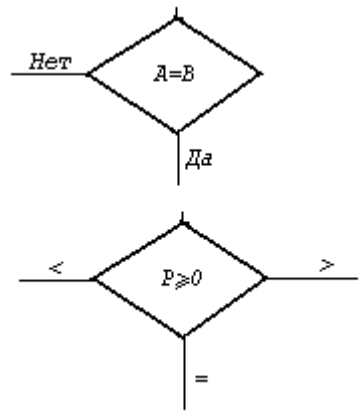
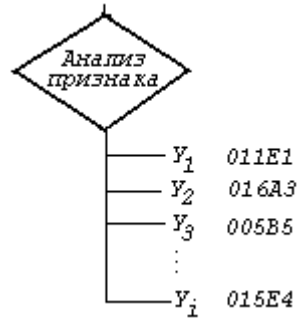
2.1. Применения символов должно соответствовать указанному в таблице.

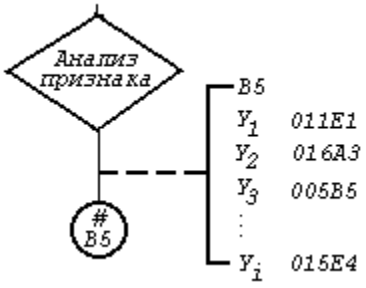
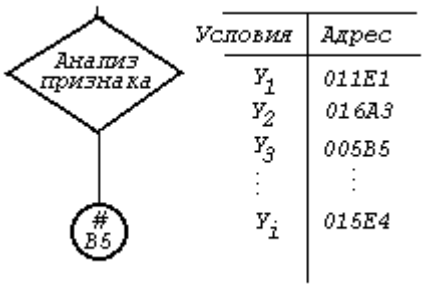
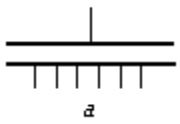
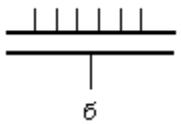
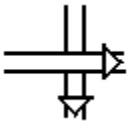
Фрагмент схемы	**Содержание обозначения**	**Правила применения**
	Возможные варианты обозначения символов в схемах: *B2, B3, C3* - координаты зоны листа, в которой размещен символ	Координаты зоны символа или порядковый номер проставляют в верхней части символа в разрыве его контура.
	*18, 19, 20* - порядковые номера символов на схеме	
		Допускается не проставлять координаты символов при выполнении схем от руки и при наличии координатной сетки.

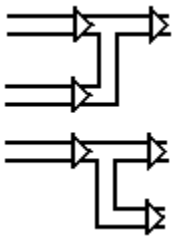
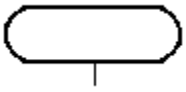
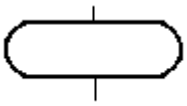
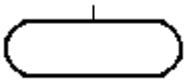
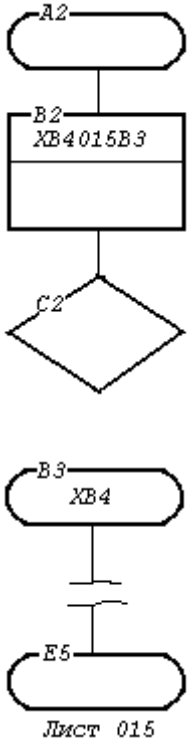
	Комментарий	Применяется, если пояснение не помещается внутри символа (для пояснения характера параметров, особенностей процесса, линий потока и др.). Комментарий записывают параллельно основной надписи. Комментарий помещают в свободном месте схемы на данном листе и соединяют с поясняемым символом.
	Соединитель: *E5, B1, A, 5* - идентификаторы соединителя в виде: буквы и цифры (координаты зоны листа)	При большой насыщенности схемы символами отдельные линии потока между удаленными друг от друга символами допускается обрывать. При этом в конце (начале) обрыва должен быть помещен символ
	буквы	

	цифры	«Соединитель».
	Межстраничный соединитель Первая строка внутри межстраничного соединителя определяет номер листа, вторая - координату символа	а) Связывание линией потока символы находятся на разных листах. **Примечание.** При изготовлении схем с помощью ЭВМ допускается указывать рядом с обрывом линии потока адресные ссылки без использования символов «Соединитель» и «Межстраничный соединитель»;
	*A3* - определяет зону на данном листе, где расположен символ «Комментарий» *010E3* - определяет номер листа и зону расположения, связываемые с символом *E3*.	б) и в случае связи некоторого символа со многими другими символами, расположенными на разных листах, на входе этого символа помещают один символ «Межстраничный соединитель», внутри которого на первой строке помещают знак # , а

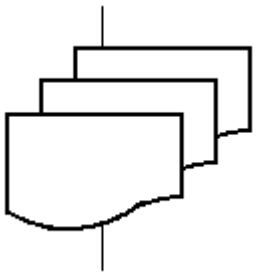
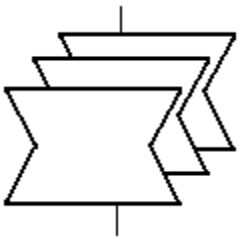
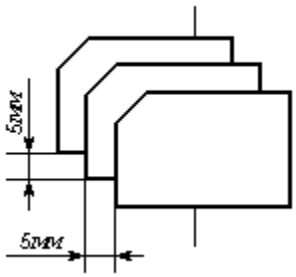
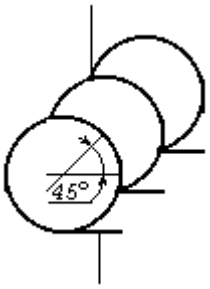
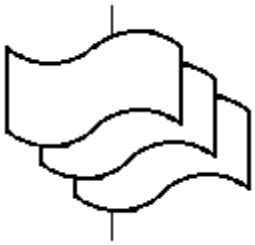
		на второй строке - координаты символа «Комментарий». Внутри символа «Комментарий» указывают номера страниц и координаты символов, связанных с поясняемым символом.
	Линии потока	Применяют для указания направления линии потока: • можно без стрелки, если линия направлена слева направо и сверху вниз; • со стрелкой - в остальных случаях.
	Излом линии под углом 90^{o}	Обозначает изменение направление потока
	Пересечение линий потока	Применяется в случае пересечения двух несвязанных потоков

	Слияние линий потока: место слияний потока обозначено точкой	Применяется в случае слияния линий потока, каждая из которых направлена к одному и тому же символу на схеме.
	место слияний потока обозначено цифрой 0	Место слияния линий потока допускается обозначать точкой или цифрой 0
		При выполнении схем на машине стрелка на линии потока выполняется прописной буквой «X» или прописной русской буквой «Х»
	Возможные варианты отображения решения: A=B, $P \geqslant 0$ - условия решений; A, B, P - параметры	При числе исходов не более трех признак условия решения (Да, Нет, =, <, >) проставляют над каждой выходящей линией потока или справа от линии потока
	y_i - условие *i*-го исхода, *011E1*, *016A3*, *005B5*, *015E4* - адреса исходов. Структура адреса имеет вид XXX XX | | координата	При числе исходов более трех условие исхода проставляется в разрыве линии потока. Адрес исхода проставляется в продолжении

	символа | номер листа схемы	условия исхода и отделяется от него пробелом;
	*B5* - знак, указывающий, что условия решения даются в виде таблицы или символа «Комментарий», расположенный на данном листе в зоне *B5*	в символе «Соединитель» указывают координату зоны, куда должна помещаться таблица или символ «Комментарий»
		в таблице (в символе «Комментарий») приводят адреса всех переходов
	Параллельные действия: начало	Применяется в случае одновременного выполнения операций, отображаемых несколькими символами
	конец	При этом в случае *а* изображается одна входная, а в случае *б* - одна выходная линия потока
	Взаимодействие материальных потоков	Применяют: при пересечении материальных

		потоков
		при объединении материальных потоков при разветвлении материальных потоков
	Начало, прерывание и конец алгоритма или программы: пуск	Символы применяют в начале схемы алгоритма или программы, в случае прерывания и в конце Внутри символа «Пуск-останов» может указываться наименование действия или идентификатор программы
	прерывание	
	останов	
	Детализация некоторой программы, представленной в данной схеме одним символом: • *XB4* - идентификатор программы; • *015* - номер листа, где проведено начало детализируемой	Применяется (в отличие от случая, когда применяется символ «Предопределенный процесс») для детализации в составе данной схемы программы. Детализируемая программа начинается и заканчивается символом «Пуск-останов».

	программы; • *B3* - координата зоны листа.	Внутри символа, посредством которого детализируется программа, проводят горизонтальную линию. В данном примере детализируемая программа представлена посредством символа «Процесс». Слева над горизонтальной линией помещается идентификатор детализируемой программы, а справа - номер листа и координата зоны, где размещен символ «Пуск-останов». Внутри символа «Пуск-останов», обозначающее начало детализируемой программы, указывается идентификатор данной программы.

	Компактное представление множества носителей данных одинакового вида: документы	Применяется, когда каждое из обозначенных множеств носителей данных обладает определенным набором свойств и имеет линии потоков одного вида и направления.
	ручные документы	
	перфокарты	
	магнитные ленты	
	перфоленты	

Приложение 3.
ГОСТ 19.003-80

УДК 003.62:681.3.06:651.7/.78:006.354 **Группа Т55**

ГОСУДАРСТВЕННЫЙ СТАНДАРТ СОЮЗА ССР

Единая система программной документации

ГОСТ

СХЕМЫ АЛГОРИТМОВ И ПРОГРАММ. ОБОЗНАЧЕНИЕ УСЛОВНЫЕ ГРАФИЧЕСКИЕ

19.003-80
Взамен
ГОСТ
19428-74

United system for program documentation.
Graphical flowchart symbols.

Постановлением Государственного комитета СССР по стандартам от 24 апреля 1980 г. [1] 1867 срок введения установлен

с 01.07 1981 г.

Настоящий стандарт распространяется на условные графические обозначения (символы) в схемах алгоритмов и программ, отображающие основные операции процесса обработки данных и программирования для систем программного обеспечения вычислительных машин, комплексов и систем независимо от их назначения и области применения.

Стандарт не распространяется на записи и обозначения, помещаемые внутри символа или рядом с ним, служащие для уточнения выполненных им функций.

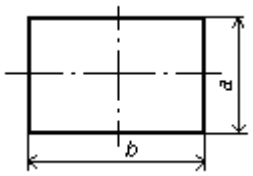
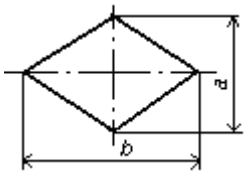
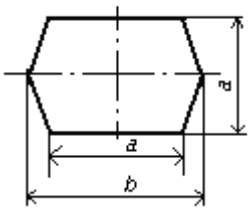
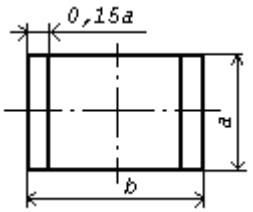
Стандарт устанавливает перечень, наименование, форму, размеры символов и отображаемые символами функции.

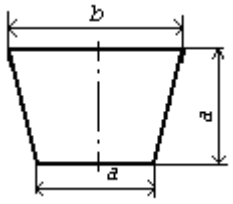
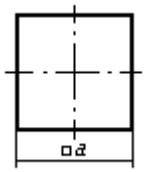
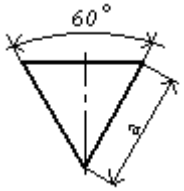
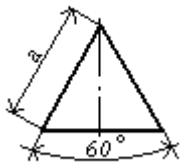
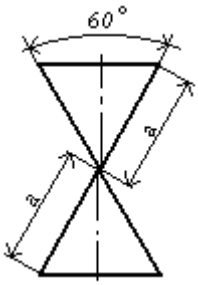
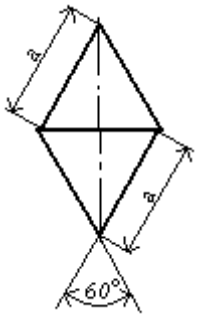
Стандарт соответствует МС ИСО 1028-73 в части обозначений символов

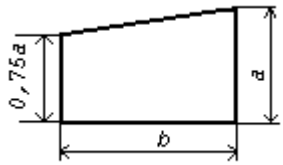
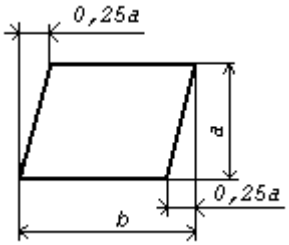
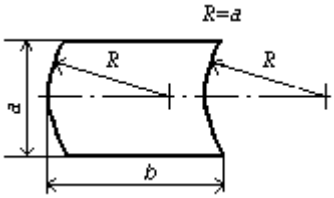
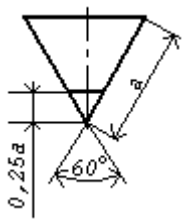
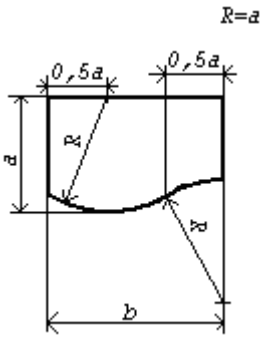
1. ПЕРЕЧЕНЬ, НАИМЕНОВАНИЕ, ОБОЗНАЧЕНИЕ СИМВОЛОВ И ОТОБРАЖАЕМЫЕ ИМИ ФУНКЦИИ

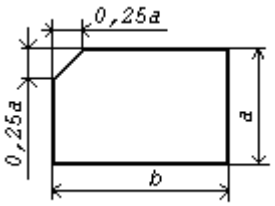
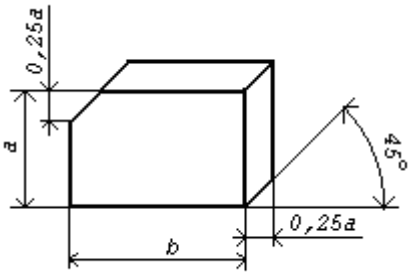
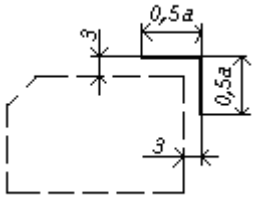
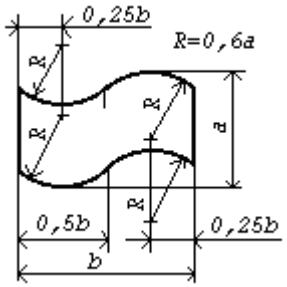
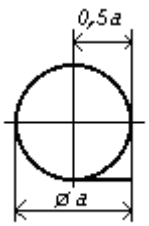
1.1. Перечень, наименование, обозначение и размеры обязательных символов и отображаемые ими функции в алгоритме и программе обработки данных должны соответствовать указанным в табл. 1.

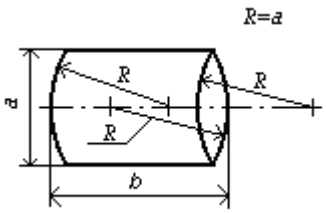
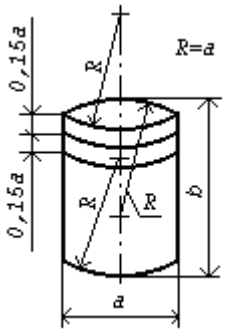
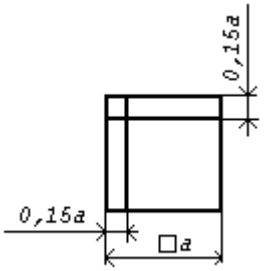
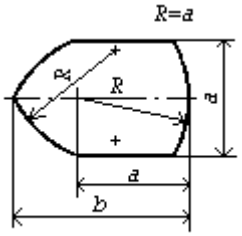
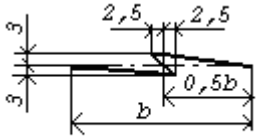
Таблица 1.

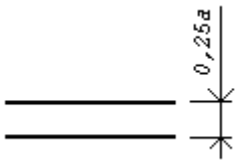
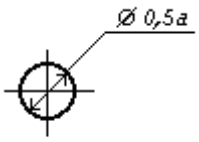
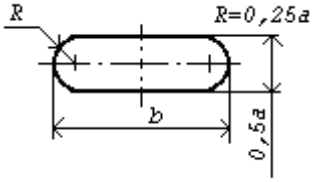
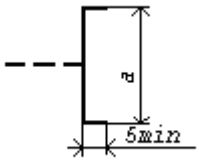
Наименование	Обозначение и размеры в мм	Функция
1. Процесс		Выполнение операций или группы операций, в результате которых изменяется значение, форма представления или расположение данных
2. Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
3. Модификация		Выполнение операций, меняющих команды или группу команд, изменяющих программу
4. Предопределенный процесс		Использование ранее созданных и отдельно описанных алгоритмов или программ

5. Ручная операция		Автономный процесс, выполняемый вручную или при помощи неавтоматически действующих средств
6. Вспомогательная операция		Автономный процесс, выполняемый устройством, не управляемым непосредственно процессором
7. Слияние		Объединение двух или более множеств в единое множество
8. Выделение		Удаление одного или нескольких множеств из единого множества
9. Группировка		Объединение двух или более множеств с выделением нескольких других множеств
10. Сортировка		Упорядочение множества по заданным признакам

11. Ручной ввод		Ввод данных вручную при помощи неавтономных устройств с клавиатурой, набором переключателей, кнопок
12. Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод)
13. Неавтономная память		Ввод-вывод данных в случае использования запоминающего устройства, управляемого непосредственно процессором
14. Автономная память		Ввод-вывод данных в случае использования запоминающего устройства, не управляемого непосредственно процессором
15. Документ		Ввод-вывод данных, носителем которых служит бумага

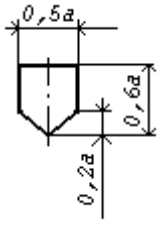
16. Перфокарта		Ввод-вывод данных, носителем которых служит перфокарта
17. Колода перфокарт		Отображение набора перфокарт
18. Файл		Представление организованных на основе общих признаков данных, характеризующих в совокупности некоторый объект обработки данных. Символ используется в сочетании с символами конкретных носителей данных, выполняющих функции ввода-вывода
19. Перфолента		Ввод-вывод данных, носителем которых служит перфолента
20. Магнитная лента		Ввод-вывод данных, носителем которых служит магнитная лента

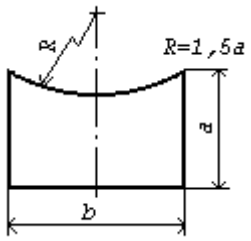
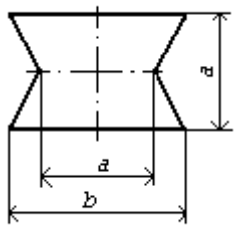
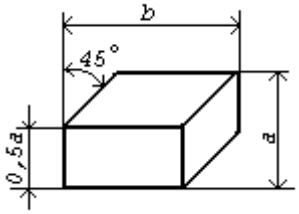
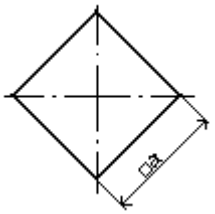
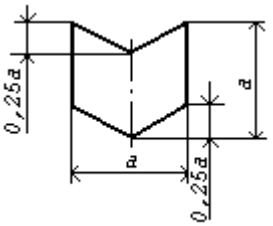
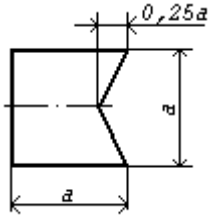
21. Магнитный барабан		Ввод-вывод данных, носителем которых служит магнитный барабан
22. Магнитный диск		Ввод-вывод данных, носителем которых служит магнитный диск
23. Оперативная память		Ввод-вывод данных, носителем которых служит магнитный сердечник
24. Дисплей		Ввод-вывод данных, если непосредственно подключенное к процессу устройство воспроизводит данные и позволяет оператору ЭВМ вносить изменения в процессе их обработки
25. Канал связи		Передача данных по каналам связи
26. Линия потока	———	Указание последовательности между символами

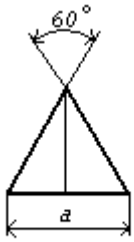
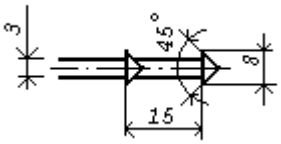
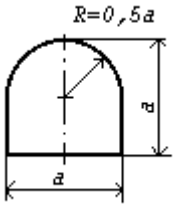
27. Параллельные действия		Начало или окончание двух и более одновременно выполняемых операций
28. Соединитель		Указание связи между прерванными линиями потока, связывающими символами
29. Пуск - останов		Начало, конец, прерывание процесса обработки данных или выполнения программы
30. Комментарий		Связь между элементом схемы и пояснением

1.2. Перечень, наименование, обозначение и размеры рекомендуемых символов и отображаемые ими функции в алгоритме и программе обработки данных должны соответствовать указанным в табл. 2.

Таблица 2

Наименование	**Обозначение и размеры в мм**	**Функция**
1. Межстраничный соединитель		Указание связи между разъединенными частями схем алгоритмов и программ, расположенных на разных листах

2. Магнитная карта		Ввод-вывод данных, носителем которых служит магнитная карта
3. Ручной документ		Формирование документа в результате выполнения ручных операций
4. Архив		Хранение комплекта упорядоченных носителей данных в целях повторного применения
5. Автономная обработка		Преобразование исходных данных в результате выполнения автономных операций
6. Расшифровка		Считывание с носителя данных, перекодирование и печать на том же или другом носителе данных в результате выполнения автономной операции
7. Кодирование		Нанесение кодированной информации на носитель в результате выполнения автономной операции

8. Копирование		Образование копии носителя в результате выполнения автономной операции
9. Транспортирование носителей		Перемещение носителей данных при помощи транспортных средств или курьером
10. Материальный поток		Указание последовательности операций в технологическом процессе изготовления предметов труда, направление их перемещения
11. Источник (приемник) данных		Отправитель или получатель данных

2. СООТНОШЕНИЕ ГЕОМЕТРИЧЕСКИХ ЭЛЕМЕНТОВ СИМВОЛОВ

2.1. Размер a должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер a на число, кратное 5. Размер b равен 1,5a.

Примечание. При ручном выполнении схем алгоритмов и программ для обязательных символов 1-5, 11, 12, 16, 29 и рекомендуемых символов 3 и 4 допускается устанавливать b равным 2a. Обязательные символы 7-10, 14 и рекомендуемый символ 8 допускается представлять в виде равнобедренного прямоугольного треугольника с катетом a.

2.2. При выполнении условных графических обозначений автоматизированным способом размеры геометрических элементов

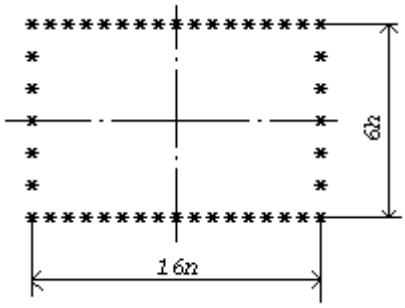
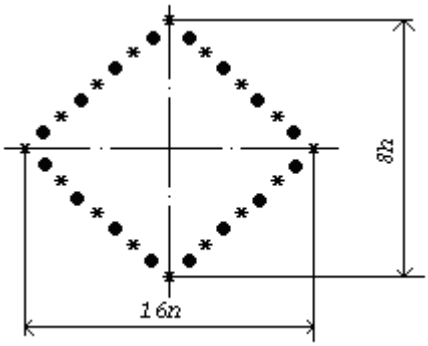
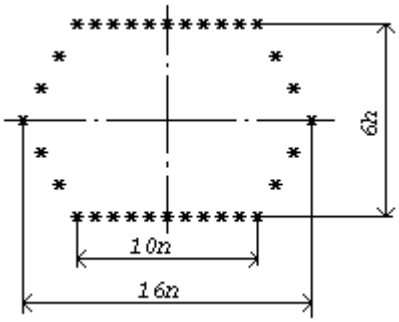
символов округляются до значений, определяемых техническими возможностями используемых устройств.

В справочном приложении приведены некоторые символы, выполненные с помощью печатающих устройств, где

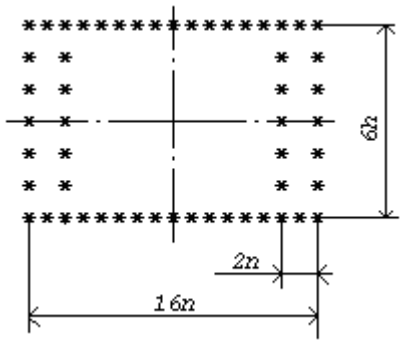
- h - шаг печатающего устройства по вертикали,
- n - шаг печатающего устройства по горизонтали.

ПРИЛОЖЕНИЕ
Справочное

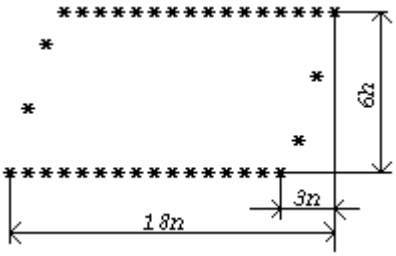
СИМВОЛЫ, ВЫПОЛНЕННЫЕ С ПОМОЩЬЮ ПЕЧАТАЮЩИХ УСТРОЙСТВ

Наименование	**Обозначение**
1. Процесс	
2. Решение	
3. Модификация	

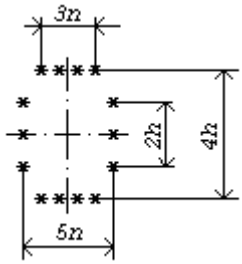
4. Предопределенный процесс



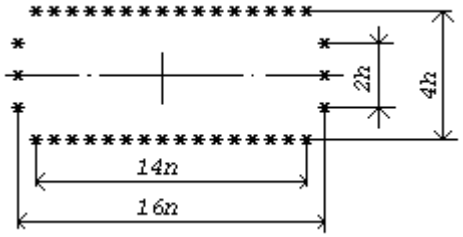
5. Ввод-вывод



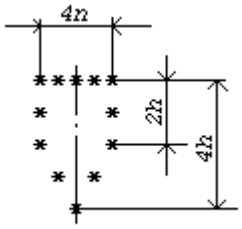
6. Соединитель



7. Пуск - останов



8. Межстраничный соединитель



9. Линия потока

Переиздание. Ноябрь 1987 г.

Приложение 4.
Пример «каркасного» приложения к заданию 1

{Основная программа}
program A;

```
  uses b,c,d,e; {Будем использовать модули B, C, D и E}
  var n:byte; {Переменная для хранения номера решаемой
задачи}

  begin

    repeat {Основной цикл работы}

      write('Task number (0-5)?:'); {Вводим номер задачи, 0 -
                                     выход из программы}
      readln(n);

      case n of {Вызываем процедуру в соответствии с введенным
                 номером задачи}
        1:p1;
        2:p2;
        3:p3;
        4:p4;
        5:p5
      end

    until n=0 {Повторяем цикл пока не будет введен 0}

  end.
```

{Модуль B содержит процедуры для решения задач 1 и 2 (процедуры P1 и P2 соответственно)}
unit B;

```
  interface {В интерфейсной части перачисляем заголовки
             процедур, доступных извне модуля}
    procedure P1;
```

```
  procedure P2;

 implementation {В секции реализации описываем процедуры
                 полностью}
  procedure P1; {Процедура решения задачи 1}
   var x:real; {Переменная – исходное данное для задачи 1}

   begin

    write('Hello from P1'); {Сообщаем о входе в процедуру}

    write('x=?'); {Вводим X}
    readln(x);
    writeln('x^2=',x*x:6); {Вычисляем и выводим результат}
    readln

   end;

  procedure P2; {Процедура решения задачи 2}
   begin
    write('Hello from P2'); {Сообщаем о входе в процедуру}
    readln
   end;

end.
```

{Модуль C содержит процедуру для решения задачи 3 (процедура P3)}

```
unit C;

 interface {В интерфейсной части перачисляем заголовки
                 процедур, доступных извне модуля}
  procedure P3;

 implementation {В секции реализации описываем процедуры
                 полностью}
  procedure P3; {Процедура решения задачи 3}
   begin
```

```
    write('Hello from P3'); {Сообщаем о входе в процедуру}
    readln
  end;

end.
```

{Модуль D содержит процедуру для решения задачи 4 (процедура P4)}

```
unit D;

 interface {В интерфейсной части перачисляем заголовки
                    процедур, доступных извне модуля}
  procedure P4;

 implementation {В секции реализации описываем процедуры
                    полностью}
  uses f; {Будем использовать модуль F}

  procedure P4; {Процедура решения задачи 4}
   begin
    write('Hello from P4'); {Сообщаем о входе в процедуру}
    readln;
    f1; {Вызываем вспомогательные процедуры F1 и F2 из
                                        модуля F}
    f2
  end;

end.
```

{Модуль E содержит процедуру для решения задачи 5 (процедура P5)}

```
unit E;
 interface {В интерфейсной части перачисляем заголовки
                    процедур, доступных извне модуля}
  procedure P5;

 implementation {В секции реализации описываем процедуры
                    полностью}
```

```
  uses f; {Будем использовать модуль F}

  procedure P5; {Процедура решения задачи 5}
    begin
      write('Hello from P5'); {Сообщаем о входе в процедуру}
      readln;
      f3; {Вызываем вспомогательные процедуры F3 и F4 из
                                                  модуля F}
      f4
    end;

end.
```

{Модуль F содержит две вспомогательные процедуры F1 и F2 для решения задачи 4 (вызываются из процедуры P4), а также две вспомогательные процедуры F3 и F4 для решения задачи 5 (вызываются из процедуры P5)}

```
unit F;

  interface {В интерфейсной части перачисляем заголовки
                        процедур, доступных извне модуля}
    procedure f1;
    procedure f2;
    procedure f3;
    procedure f4;

  implementation {В секции реализации описываем процедуры
                        полностью}
    procedure f1;
      begin
        write('Hello from f1'); {Сообщаем о входе в процедуру}
        readln
      end;

    procedure f2;
      begin
        write('Hello from f2'); {Сообщаем о входе в процедуру}
        readln
```

```
  end;

 procedure f3;
  begin
   write('Hello from f3'); {Сообщаем о входе в процедуру}
   readln
  end;

 procedure f4;
  begin
   write('Hello from f4'); {Сообщаем о входе в процедуру}
   readln
  end;

end.
```

Приложение 5.
Пример «каркасного» приложения к заданию 2

```
{Основная программа}
program A;

  uses b, c, d, e, g, crt {Будем использовать модули B, C, D, E, G
                           и CRT (при использовании FreePascal
                           лучше использовать PTCCRT)}
  var n:byte; {Переменная для хранения номера решаемой
задачи}

  begin

{Оформляем экран в целом}
    textbackground(lightgray); {Устанавливаем светлосерый цвет
                                фона (lightgray – константа,
                                описанная в модуле CRT)}
    clrscr; {Очищаем экран и заливаем его цветом фона}
    textcolor(white); {Устанавливаем белый цвет шрифта (этим
                       цветом будут рисоваться рамки,
                       white – константа, описанная в модуле CRT)}
    frame1; {Вызываем процедуру рисования наружной рамки}
    frame2; {Вызываем процедуру рисования внутренней рамки}
    menushow; {Вызываем процедуру рисования меню}

    repeat {Основной цикл работы}

      n:=menuselect; {Вызываем процедуру работы с меню
                      (выбора необходимого пункта в
                      соответствии с номером
задачи), номер выбранного пункта меню
помещаем в N}

{Оформляем экран для решения задачи}
      window(3,4,78,22); {Устанавливаем окно на рабочую
                          область экрана, ограниченную
                          внутренней рамкой, в
```

```
пределах которой будет происходить диалог с пользователем (ввод исходных данных и вывод результатов)}
        textbackground(lightblue); {Устанавливаем светлосиний цвет фона (lightblue – константа, описанная в модуле CRT)}
        clrscr; {Очищаем рабочее окно и заливаем его цветом фона}
        textcolor(yellow); {Устанавливаем желтый цвет шрифта
                            (этим цветом будут отображаться
                            символы в диалоге с пользователем, yellow
                            – константа, описанная в модуле CRT)}

        case n of {Вызываем процедуру в соответствии с
                   полученным номером задачи (номером
                   выбранного пункта меню)}
          1:p1;
          2:p2;
          3:p3;
          4:p4;
          5:p5;
        end;

        window(1,1,80,24)

      until n=0; {Повторяем цикл пока не будет получен 0 (выбран
                  пункт меню с номером 0 – завершение работы
                  программы)}

    {Перед завершением возвращаем стандартные настройки}
      textbackground(black); {Устанавливаем черный цвет фона
                              (black – константа,описанная в
                              модуле CRT)}
      textcolor(white); {Устанавливаем белый цвет шрифта (white
–                     константа, описанная в модуле CRT)}
      clrscr {Очищаем рабочее окно и заливаем его цветом фона}
```

```
end.
```

{Модуль B содержит процедуры для решения задач 1 и 2 (процедуры P1 и P2 соответственно)}

```
unit B;

interface {В интерфейсной части перачисляем заголовки
                процедур, доступных извне модуля}
  procedure P1;
  procedure P2;

implementation {В секции реализации описываем процедуры
                        полностью}
  procedure P1; {Процедура решения задачи 1}
    var x:real; {Переменная – исходное данное для задачи 1}

    begin

      write('Hello from P1'); {Сообщаем о входе в процедуру}

      write('x=?'); {Вводим X}
      readln(x);
      writeln('x^2=',x*x:6); {Вычисляем и выводим результат}
      readln

    end;

  procedure P2; {Процедура решения задачи 2}
    begin
      write('Hello from P2'); {Сообщаем о входе в процедуру}
      readln
    end;

end.
```

{Модуль C содержит процедуру для решения задачи 3 (процедура P3)}

```
unit C;
```

```
  interface {В интерфейсной части перачисляем заголовки
                        процедур, доступных извне модуля}
    procedure P3;

  implementation {В секции реализации описываем процедуры
                        полностью}
    procedure P3; {Процедура решения задачи 3}
      begin
        write('Hello from P3'); {Сообщаем о входе в процедуру}
        readln
      end;

end.
```

{Модуль D содержит процедуру для решения задачи 4 (процедура P4)}

```
unit D;

  interface {В интерфейсной части перачисляем заголовки
                        процедур, доступных извне модуля}
    procedure P4;

  implementation {В секции реализации описываем процедуры
                        полностью}
    uses f; {Будем использовать модуль F}

    procedure P4; {Процедура решения задачи 4}
      begin
        write('Hello from P4'); {Сообщаем о входе в процедуру}
        readln;
        f1; {Вызываем вспомогательные процедуры F1 и F2 из
                                                модуля F}
        f2
      end;

end.
```

{Модуль E содержит процедуру для решения задачи 5 (процедура P5)}

```
unit E;
 interface {В интерфейсной части перачисляем заголовки
                          процедур, доступных извне модуля}
  procedure P5;

 implementation {В секции реализации описываем процедуры
                          полностью}
  uses f; {Будем использовать модуль F}

  procedure P5; {Процедура решения задачи 5}
   begin
    write('Hello from P5'); {Сообщаем о входе в процедуру}
    readln;
    f3; {Вызываем вспомогательные процедуры F3 и F4 из
                                                модуля F}
    f4
   end;

end.
```

{Модуль F содержит две вспомогательные процедуры F1 и F2 для решения задачи 4 (вызываются из процедуры P4), а также две вспомогательные процедуры F3 и F4 для решения задачи 5 (вызываются из процедуры P5)}

```
unit F;

 interface {В интерфейсной части перачисляем заголовки
                          процедур, доступных извне модуля}
  procedure f1;
  procedure f2;
  procedure f3;
  procedure f4;

 implementation {В секции реализации описываем процедуры
                          полностью}
  procedure f1;
```

```
  begin
    write('Hello from f1'); {Сообщаем о входе в процедуру}
    readln
  end;

procedure f2;
  begin
    write('Hello from f2'); {Сообщаем о входе в процедуру}
    readln
  end;

procedure f3;
  begin
    write('Hello from f3'); {Сообщаем о входе в процедуру}
    readln
  end;

procedure f4;
  begin
    write('Hello from f4'); {Сообщаем о входе в процедуру}
    readln
  end;

end.
```

{Модуль G содержит процедуры для рисования наружной и внутренней рамок (frame1 и frame2 соответственно), формирования меню (menushow) и функцию menuselect для выбора интересующего пункта меню, возвращающую в качестве результата номер выбранного пункта}

```
unit g;

 interface {В интерфейсной части перачисляем заголовки
                процедур и функций, доступных извне модуля}
  procedure frame1;
  procedure frame2;
  procedure menushow;
  function menuselect:byte;
```

```
implementation {В секции реализации описываем процедуры и
                                        функции полностью}
  uses crt; {Будем использовать модуль CRT (при использовании
                                        FreePascal лучше использовать PTCCRT)}

 {Массив строковых констант MENUTEXTS содержит надписи
на пунктах меню}
    const menutexts:array [0..5] of string [10]=
                    ('Выход','Задача 1','Задача 2',
                    'Задача 3','Задача 4','Задача 5');
    var n:byte; {Переменная для хранения номера выделенного
                                        (активного) пункта меню}
        menux:array [0..5] of byte; {Массив для хранения
                                        координат пунктов меню}

    procedure frame1; {Процедура рисования наружной рамки}

      var i:byte; {Вспомогательная переменная}

      begin

        for i:=2 to 23 do {В цикле символами псевдографики
рисуем                          вертикальные (левую и правую)
                                                линии
рамки}
          begin
            gotoxy(1,i);write(#179);
            gotoxy(80,i);write(#179)
          end;

        for i:=2 to 79 do {В цикле символами псевдографики
рисуем                          горизонтальные (верхнюю и
нижнюю)                                         линии
рамки}
          begin
            gotoxy(i,1);write(#196);
            gotoxy(i,24);write(#196)
```

```
      end;

    gotoxy(1,1);write(#218); {Символами псевдографики
                              соединяем вертикальные и
                              горизонтальные линии в углах
                                                  рамки}
    gotoxy(1,24);write(#192);
    gotoxy(80,1);write(#191);
    gotoxy(80,24);write(#217)

  end;

procedure frame2; {Процедура рисования внутренней рамки}

 var i:byte; {Вспомогательная переменная}

 begin

   for i:=4 to 22 do {В цикле символами псевдографики рисуем
                      вертикальные (левую и правую)
                                          линии рамки}
     begin
       gotoxy(2,i);write(#186);
       gotoxy(79,i);write(#186)
     end;

   for i:=3 to 78 do {В цикле символами псевдографики рисуем
                      горизонтальные (верхнюю и нижнюю)
                                          линии рамки}
     begin
       gotoxy(i,3);write(#205);
       gotoxy(i,23);write(#205)
     end;

    gotoxy(2,3);write(#201); {Символами псевдографики
                              соединяем вертикальные и
                              горизонтальные линии в углах
                                                  рамки}
```

```
      gotoxy(2,23);write(#200);
      gotoxy(79,3);write(#187);
      gotoxy(79,23);write(#188)

  end;

  procedure menushow; {Процедура рисования меню}

  var i:byte; {Вспомогательная переменная}

  begin

{Выводим на экран все пункты меню}
    textbackground(lightgray); {Символы будем выводить на
                                светлосером фоне}
    textcolor(black); {Сами символы будут черными}
    for i:=0 to 5 do {В цикле смещаем курсор до координат
                      начала пункта и выводим надписи пунктов
                                                меню}
      begin
        gotoxy(menux[i],2);write(menutexts[i])
      end;

{Перерисовываем выделенный (активный ) пункт меню}
    textbackground(lightcyan); {Символы будем выводить на
                                светлобирюзовом фоне}
    textcolor(white); {Сами символы будут белыми}
    gotoxy(menux[n],2);write(menutexts[n]) {Смещаем курсор до
                                координат начала
                               пункта и выводим надпись}

  end;

  function menuselect:byte; {Функциа выбора пункта меню, в
                             качаестве результата возвращает
                            номер пункта, выбранного пользователем}
  var c:char; {Переменная для хранения кода нажатой
                            клавиши(стрелки влево/вправо или Enter)}
```

```
begin

  {Основной цикл работы, в котором выделение перемещается
по пунктам меню (происходит смена активного пункта) в
соответствии с нажатиями стрелок влево/вправо пользователем}
    Repeat

      c:=readkey; {Вводим код нажатого символа}
      case c of {В соответствии с кодом пулученного символа
                 изменяем номер выделенного (активного)
                                                 пункта
меню}
        #75: if n=0 {Если была нажата стрелка влево –
                     уменьшаем номер активного пункта
                     (уменьшение происходит циклически, после
                                                 0 идет 5)}
               then n:=5
               else n:=n-1;
        #77: if n=5 {Если была нажата стрелка вправо –
                     увеличиваем номер активного пункта
                     (увеличение происходит циклически,
после                                            5 идет
0)}
               then n:=0
               else n:=n+1
      end;

      menushow {Перерисовываем меню, чтобы показать новое
                                 положение активного пункта меню}

    until c=#13; {Цикл заканчивается при нажатии на Enter}

    menuselect:=n {В качестве результата возвращаем
текущий                          номер активного пункта меню}

  end;
```

```
{Секция инициализации модуля}
Begin

{Рассчитываем координаты начала пунктов меню и размещаем их в массиве координат MENUX}
  menux[0]:=5; {Первый (самый левый) понкт меню начнется с пятой позиции строки}
  for n:=1 to 5 do {В цикле рссчитываем координаты (номера позиций) остальных пунктов}
   menux[n]:=menux[n-1]+length(menutexts[n-1])+4;  {позиция очередного пункта есть сумма номера позиции начала предыдущего пункта плюс длина текста надписи предыдущего пункта плюс 4 позиции на интервал между пунктами меню}

  n:=0 {Задаем начальный номер активного пункта меню: при запуске программы активным будет самый левый пункт}

end.
```

Приложение 6.
Пример «каркасного» приложения к заданию 3

{Основная программа (рассчитана на компиляцию в среде Free Pascal)}

```
program main;

uses uapp; {Будем использовать процедуры из модуля UAPP}

begin

  init; {Инициализируем графику и оформляем экран}
  run; {Основная рабочая процедура}
  done {Очищаем экран и возвращаем настройки по умолчанию}

end.
```

{Модуль UAPP содержит процедуры инициализации графики (создания графического окна) и оформления экрана, основную рабочую процедуру для выбора пункта меню и управления графическим объектом (фигурой), а также завершающую процедуру для очистки и уничтожения графического окна}

```
unit uapp;

interface {В интерфейсной части перачисляем заголовки
                процедур, доступных извне модуля}

  procedure init;
  procedure run;
  procedure done;

implementation {В секции реализации описываем процедуры и
                функции полностью}
uses ptcgraph, umenu, uctrl, uchange, uobj; {Будем использовать
модули PTCGRAPH, UMENU, UCTRL, UCHANGE, UOBJ
}
  procedure init; {Процедура инициализации графики: создает
                графическое окно, рисует меню и
```

```
{графический объект (фигуру)}
        var
         grDriver, grMode, ErrCode: Integer; {Вспомогательные
                                              переменные для
                                              инициализации графики}

        begin

         grDriver := vga;  grmode := vgahi; {Задаем режимы работы
                                             графики}
         InitGraph(grDriver, grMode,''); {Пытаемся создать
                                          графическое окно}
         ErrCode := GraphResult; {Получаем код успешности создания
                                  графического окна}
         if ErrCode <> grOk {Если при создании окна возникла ошибка,
                             выводим сообщение о ней и
завершаем выполнение программы}
           then begin writeln('InitGraph error'); halt end;

         myobj_init; {Если окно создано – инициализируем и рисуем
                      фигуру}
         menu_init {Инициализируем и рисуем меню}

        end;

       procedure run; {Основная рабочая процедура, позволяющая
                       пользователю выбрать нужнй пункт меню и
                       вызывающая процедуру для выполнения
                       соответствующей операции}

         var n:integer; {Переменная для хранения номера выбранного
пункта меню}

         begin

           repeat {Основной цикл работы}
```

```
        menu_show; {Показываем меню}
        n:=menu_select; {Даем возможность пользователю
                         выбрать нужный пункт, номер
выбранного пункта
помещаем в N}
        menu_hide; {Скрываем меню (фигура будет двигаться в
                                   пределах всего окна)}

        case n of {Вызываем процедуру, выполняющую операцию в
                   соответствии с выбранным пунктом}
          1:move;
          2:rotate;
          3:move_control;
          4:rotate_control;
        end;

        myobj_hide {Скрываем фигуру, чтобы не возникло
                    возможного наложения меню и фигуры}

      until n=5 {Цикл будет выполняться, пока пользователь не
выберет пункт 5 – выход}

    end;

  procedure done; {Завершающая процедура}

    begin

     cleardevice; {Очищаем экран}
     closegraph {Закрываем графическое окно}

    end;

end.
```

{Модуль UCHANGE содержит процедуры изменения положения фигуры, а также ее инициализации}

```
unit uchange;

interface {В интерфейсной части перачисляем заголовки
           процедур, доступных извне модуля}

    procedure myobj_up;
    procedure myobj_down;
    procedure myobj_left;
    procedure myobj_right;
    procedure myobj_up_right;
    procedure myobj_up_left;
    procedure myobj_down_right;
    procedure myobj_down_left;
    procedure myobj_rotate_right;
    procedure myobj_rotate_left;
    procedure myobj_init;

implementation {В секции реализации описываем процедуры и
                функции полностью}

 uses uobj, ptcgraph; {Будем использовать модули UOBJ,
                       PTCGRAPH}

 const dy=5; dx=5; dangle=0.1; {Константы, определяющие шаг
                изменения координат и
угла            поворота фигуры}

 procedure myobj_up; {Процедура смещения фигуры на один
                      шаг вверх}
  begin

   myobj_hide; {Скрываем фигуру по старым координатам}

   y:=y-dy; {Изменяем координаты}
   if y<0 then y:=y+getmaxy; {Циклически в пределах окна}

   myobj_show; {Рисуем фигуру по новым координатам}
```

```
    end;

  procedure myobj_down; {Процедура смещения фигуры на один шаг
вниз}
    begin

      myobj_hide; {Скрываем фигуру по старым координатам}

      y:=y+dy; {Изменяем координаты}
      if y>getmaxy then y:=y-getmaxy; {Циклически в пределах
окна}

      myobj_show {Рисуем фигуру по новым координатам}

    end;

  procedure myobj_right; {Процедура смещения фигуры на один
                                                шаг вправо}
    begin

      myobj_hide; {Скрываем фигуру по старым координатам}

      x:=x+dx; {Изменяем координаты}
      if x>getmaxx then x:=x-getmaxx; {Циклически в пределах
окна}

      myobj_show; {Рисуем фигуру по новым координатам}

    end;

  procedure myobj_left; {Процедура смещения фигуры на один
                                                шаг
влево}
    begin

      myobj_hide; {Скрываем фигуру по старым координатам}
```

```
    x:=x-dx; {Изменяем координаты}
    if x<0 then x:=x+getmaxx; {Циклически в пределах окна}

    myobj_show {Рисуем фигуру по новым координатам}

  end;

procedure myobj_up_right; {Процедура смещения фигуры на
                                один шаг вверх и
вправо}
  begin

    myobj_hide; {Скрываем фигуру по старым координатам}

    y:=y-dy; {Изменяем координаты}
    if y<0 then y:=y+getmaxy; {Циклически в пределах окна}
    x:=x+dx; {Изменяем координаты}
    if x>getmaxx then x:=x-getmaxx; {Циклически в пределах
окна}

    myobj_show {Рисуем фигуру по новым координатам}

  end;

procedure myobj_up_left; {Процедура смещения фигуры на
                                один шаг вверх и влево}
  begin

    myobj_hide; {Скрываем фигуру по старым координатам}

    y:=y-dy; {Изменяем координаты}
    if y<0 then y:=y+getmaxy; {Циклически в пределах окна}
    x:=x-dx; {Изменяем координаты}
    if x<0 then x:=x+getmaxx; {Циклически в пределах окна}

    myobj_show {Рисуем фигуру по новым координатам}

  end;
```

```
procedure myobj_down_right; {Процедура смещения фигуры
                                    на один шаг вниз и вправо}
  begin

    myobj_hide; {Скрываем фигуру по старым координатам}

    y:=y+dy; {Изменяем координаты}
    if y>getmaxy then y:=y-getmaxy; {Циклически в пределах
окна}
    x:=x+dx; {Изменяем координаты}
    if x>getmaxx then x:=x-getmaxx; {Циклически в пределах
окна}

    myobj_show {Рисуем фигуру по новым координатам}

  end;

procedure myobj_down_left; {Процедура смещения фигуры на
                                    один шаг вниз и влево}
  begin

    myobj_hide; {Скрываем фигуру по старым координатам}

    y:=y+dy; {Изменяем координаты}
    if y>getmaxy then y:=y-getmaxy; {Циклически в пределах
окна}
    x:=x-dx; {Изменяем координаты}
    if x<0 then x:=x+getmaxx; {Циклически в пределах окна}

    myobj_show {Рисуем фигуру по новым координатам}

  end;

procedure myobj_rotate_right; {Процедура вращения фигуры
                                    на один шаг вправо}
  begin
```

```
    myobj_hide; {Скрываем фигуру по старым координатам}

    angle:=angle-dangle; {Изменяем угол поворота}
    if angle <0 then angle:=angle+6.28; {Циклически в пределах
                                        диапазона 0 – 6,28}

    myobj_show {Рисуем фигуру по новым координатам}

  end;

 procedure myobj_rotate_left; {Процедура вращения фигуры на
                                        один шаг влево}
  begin

    myobj_hide; {Скрываем фигуру по старым координатам}

    angle:=angle+dangle; {Изменяем угол поворота}
    if angle>6.28 then angle:=angle-6.28; {Циклически в пределах
                                        диапазона 0 – 6,28}

    myobj_show {Рисуем фигуру по новым координатам}

  end;

 procedure myobj_init; {Процедура инициализации фигуры}

  begin

    x:=getmaxx div 2; {Координата X – центр окна}
    y:=getmaxy div 2; {Координата Y – центр окна}
    angle:=0; {Начальный угол поворота равен нулю}

    myobj_show {Рисуем фигуру в центре окна}

  end;

end.
```

{Модуль UCTRL содержит процедуры, выполняющие требуемые операции над фигурой}

unit uctrl;

interface {*В интерфейсной части перачисляем заголовки процедур, доступных извне модуля}*

```
  procedure move;
  procedure rotate;
  procedure move_control;
  procedure rotate_control;
```

implementation *{В секции реализации описываем процедуры и функции полностью}*

uses uchange, uobj, ptccrt; {*Будем использовать модули UCHANGE, UOBJ иPTCCRT}*

```
  const delay_time=100; {Канстанта, ограничивающая скорость движения/вращения фигуры}
```

procedure move; *{Процедура движения фигуры}*

```
    var ch:char; {Переменная для хранения кода нажатого
```

символа}

```
    begin
      myobj_show; {Показываем фигуру}
      repeat {Основной цикл работы}
        ch:=readkey; {Дожидаемся нажатия на клавишу, код
                      символа помещаем в CH}
```

```
      case ch of {В зависимости от кода введенного символа
                  вызываем процедуру перемещения в нужном
                                                  направлении}
        '1':myobj_down_left;
        '2':myobj_down;
        '3':myobj_down_right;
        '4':myobj_left;
        '6':myobj_right;
        '7':myobj_up_left;
        '8':myobj_up;
        '9':myobj_up_right;

      end;

    until ch=#27 {Выход из цикла движения происходит по
                                  нажатию клавиши ESC (Escape)}
  end;

 procedure rotate; {Процедура вращения фигуры}

  var ch:char; {Переменная для хранения кода нажатого
символа}

  begin

    myobj_show; {Показываем фигуру}

    repeat {Основной цикл работы}

      ch:=readkey; {Дожидаемся нажатия на клавишу, код
                                  символа помещаем в CH}

      case ch of {В зависимости от кода введенного символа
                  вызываем процедуру поворота в нужном
направлении}
        #75:myobj_rotate_left;
```

```
        #77:myobj_rotate_right;
      end;
    until ch=#27 {Выход из цикла вращения происходит по
                                нажатию клавиши ESC (Escape)}
  end;
procedure move_control; {Процедура управления движением
фигуры}
  var ch:char; {Переменная для хранения кода нажатого
символа}
  begin
    myobj_show; {Показываем фигуру}
    ch:='5'; {В CH помещаем код символа, при котором фигура
                                                    не
движется}
    repeat {Основной цикл работы}
      if keypressed then ch:=readkey; {Проверяем, была ли
нажата                          клавиша, если да –
                          помещаем код символа в
CH}
      case ch of {В зависимости от кода введенного символа
                  вызываем процедуру перемещения в нужном
                                              направлении}
        '1':myobj_down_left;
        '2':myobj_down;
        '3':myobj_down_right;
```

```
        '4':myobj_left;
        '6':myobj_right;
        '7':myobj_up_left;
        '8':myobj_up;
        '9':myobj_up_right;

      end;

      delay(delay_time) {после очередного шага перемещения
                         приостанавливаем движение на
                         DELAY_TIME миллисекунд}

      until ch=#27 {Выход из цикла управления движением
                    происходит по нажатию клавиши ESC
    (Escape)}

    end;

  procedure rotate_control; {Процедура управления вращением
  фигуры}

    var ch:char; {Переменная для хранения кода нажатого
  символа}

    begin

      myobj_show; {Показываем фигуру}

      ch:='5'; {В CH помещаем код символа, при котором фигура
                не движется}

      repeat {Основной цикл работы}
```

```
        if keypressed then ch:=readkey; {Проверяем, была ли нажата
                                        клавиша, если да –
                                        помещаем код символа в
CH}

          case ch of {В зависимости от кода введенного символа
                      вызываем процедуру поворота в нужном
                                        направлении}
            #75:myobj_rotate_left;
            #77:myobj_rotate_right;

          end;

          delay(delay_time) {после очередного шага поворота
                      приостанавливаем вращение на
                          DELAY_TIME миллисекунд}

        until ch=#27 {Выход из цикла управления вращением
                      происходит по нажатию клавиши ESC
                                        (Escape)}

      end;

    end.
```

{Модуль UMENU содержит процедуры для работы с меню}
unit umenu;

```
interface {В интерфейсной части перачисляем заголовки
           процедур и функций, доступных извне модуля}

    procedure menu_show;
    procedure menu_hide;
    function menu_select : integer;
    procedure menu_init;

implementation {В секции реализации описываем процедуры и
                функции полностью}
```

```
uses ptcgraph, ptccrt; {Будем использовать модули
                        PTCGRAPH и PTCCRT}

var {Описываем переменные}
    texts : array[1..5] of string; {Массив для хранения надписей
                                    пунктов меню}
    x,y : array[1..5] of integer; {Массивы для хранения координат
X                                   и Y пунктов меню}

    activenum : integer; {Номер активного (выделенного) пункта
                          меню}

procedure menu_show; {Процедура показа (рисования) меню}

  var i:integer; {Вспомогательная переменная}

  begin

    setcolor(white); {Устанавливаем белый цвет рисования, им
                      будут показаны все пункты меню}

{Рисуем пункты меню по координатам из массивов X и Y,
тексты надписей берем из массива TEXTS}
    for i:=1 to 5 do outtextxy(x[i],y[i],texts[i]);

    setcolor(yellow); {Устанавливаем желтый цвет рисования,
                       им будет показан активный пункт
                       меню}

{Перерисовываем активный пункт меню}
    outtextxy(x[activenum],y[activenum],texts[activenum])

  end;

procedure menu_hide; {Процедура скрытия (стирания) меню}

  var i:integer; {Вспомогательная переменная}
```

```
begin

  setcolor(getbkcolor());{Устанавливаем цвет рисования,
                          совпадающий с цветом фона, им
                          будут показаны все пункты меню}

{Рисуем пункты меню по координатам из массивов X и Y,
тексты надписей берем из массива TEXTS}
  for i:=1 to 5 do outtextxy(x[i],y[i],texts[i])

end;

function menu_select : integer; {Функция выбора пункта меню,
в                                качестве результата возвращает
                                 номер выбранного пункта}

  var ch : char; {Переменная для хранения кода нажатого

символа}

  begin

    repeat {Основной цикл работы}

      ch:=readkey; {Вводим символ и размещаем его код в CH}

      case ch of {В зависимости от кода ввыденного символа
                  изменяем номер активного пункта меню}

        #75:begin  {Если нажата стрелка влево – уменьшаем
номер                                          активного
пункта}
             dec(activenum);
             if activenum < 1  {Организуем циклическое изменение
                                номера активного пункта в
                                диапазоне 1 - 5}
               then
```

```
                activenum := 5
            end;
        #77:begin {Если нажата стрелка вправо – увеличиваем
                                        номер активного пункта}
            inc(activenum);
            if activenum > 5  {Организуем циклическое изменение
                                номера активного пункта в
                                        диапазоне 1 - 5}
                then
                    activenum := 1
            end;

        end;

        menu_show; {Перерисовываем меню с новым положением
                                        активного
пункта}

        until ch=#13; {Цикл выбора пункта меню заканчивается
                                нажатем на клавишу
Enter}

        menu_select:=activenum {В качестве результата функции
                                возвращаем номер активного
                                        пункта меню}

      end;

    procedure menu_init; {Процедура инициализации меню}

     var i:integer; {Вспомогательная переменная}

    begin

      activenum:=1; {При запуске программы активным будет
пункт                                   номер
1}
```

```
{Заполняем массив надписей пунктов меню}
  texts[1]:='Moving';
  texts[2]:='Rotation';
  texts[3]:='Moving control';
  texts[4]:='Rotation control';
  texts[5]:='Exit';

{Рассчитываем координаты пунктов меню и размещаем их в массовах координат X и Y}
  x[1]:=10; {Первый пункт меню будет отстоять на 10 точек
                                    от левой границы окна}
{Координаты остальных пунктов рассчитываются как координата предыдущего пункта плюс длина текста надписи предыдущего пункта плюс 10 точек на интервал между пунктами}
  for i:=2 to 5 do x[i]:=x[i-1]+textwidth(texts[i-1])+10;
{Координаты Y всех пунктов меню рассчитываются как высота текста надписи пункта плюс 5 точек на расстояние от верхней границы окна}
  for i:=1 to 5 do y[i]:=textheight(texts[1])+5;

 end;

 end.
```

{Модуль UOBJ содержит процедуры для показа и скрытия фигуры}

```
unit uobj;

interface {В интерфейсной части перачисляем заголовки
          процедур, а также переменные, доступные
                                  извне модуля}

     var  x, y : integer; angle:real; {Переменные для хранения текущих координат центра фигуры и ее угла поворота}

      procedure myobj_hide;
      procedure myobj_show;
```

```
implementation {В секции реализации описываем процедуры и
                функции полностью}

  uses ptcgraph; {Будем использовать модуль PTCGRAPH}

  const r=30; {Константа, задающая радиус окружности, в
               которую вписывается фигура}

  procedure myobj_draw(color:integer); {Процедура рисования
фигуры заданным цветом, цвет рисования задается через параметр
процедуры COLOR}

    begin

      setcolor(color); {устанвливаем цвет рисования, переданный
                        через параметр процедуры}

{Рисуем фигуру (в данном случае - линию)}
      line(round(x+r*cos(angle)),round(y-r*sin(angle)),
        round(x+r*cos(angle+6.28/2)),round(y-sin(angle+6.28/2)))

    end;

  procedure myobj_show; {Процедура показа фигуры}

    begin

      myobj_draw(white) {Вызываем процедуру рисования фигуры,
                         цвет - белый}

    end;

  procedure myobj_hide; {Процедура скрытия фигуры}

    begin

      myobj_draw(getbkcolor()){Вызываем процедуру рисования
                фигуры, цвет – совпадающий с цветом фона}
```

```
    end;
end.
```