Министерство общего и профессионального образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

УТВЕРЖДАЮ
Декан ФПМК
_____А. М. Горцев
“____” _____2000 г.

ОСНОВЫ ИСПОЛЬЗОВАНИЯ ГРАФИЧЕСКОЙ БИБЛИОТЕКИ ЯЗЫКА СИ И СИ++

Учебное пособие

Томск 2000

РАСМОТРЕНО и ОДОБРЕНО методической комиссией факультета прикладной математики и кибернетики.

ПРОТОКОЛ № от 2000 г.

Председатель комиссии
профессор Ю. И. Параев

В настоящем учебном пособии приведены сведения о работе основных функций графического режима Турбо Си и Си ++, приведены примеры использования этих функций, рассмотрены возможные ошибки, возникающие при их работе.

Учебное пособие предназначено для студентов технических и математических специальностей ВУЗов.

Составитель: Буторина Н. Б.

1. Необходимые технические сведения

Использование графики в Turbo C – это многошаговый процесс. Прежде всего, необходимо определить тип видеоадаптера. Затем устанавливается подходящий режим его работы и выполняется инициализация графической системы в выбранном режиме. Только после этого становятся доступными для использования функции графической библиотеки для построения основных графических примитивов.

Использование библиотеки графики Turbo C намного сокращает объем программирования для вывода основных графических примитивов. Turbo C "маскирует" многие технические детали управления оборудованием. Но платой за эти удобства является значительное увеличение размера exe-файла.

В самых общих чертах работа с дисплеем персонального компьютера в графическом режиме может быть представлена следующим образом. Экран дисплейного монитора представляется как набор отдельных точек – пикселов (pixel , от английского picture elements). Количество пикселов, которое можно разместить на экране, определяет размеры экрана и обычно отражается парой чисел, первое из которых показывает количество пикселов в одной строке, а второе – число строк, например, 640x350. Каждому пикселу экрана ставится в соответствие фиксированное количество битов (атрибут пиксела) в некоторой области адресного пространства центрального микропроцессора персонального компьютера. Эта область называется видеопамятью и является частью специального устройства, управляющего работой монитора – дисплейного адаптера (видеоадаптера). Видеоадаптер, в частности, осуществляет циклическое воспроизведение содержимого видеопамяти на экране монитора. При этом изображение каждого пиксела определяется текущим значением его атрибута. Такой подход получил название битовой карты – bit-mapped graphics. Программе, выполняющейся на персональном компьютере в графическом режиме, доступны для чтения/записи все пикселы видеопамяти.

В ряде случаев возможно одновременное существование в видеопамяти двух или более областей одинаковой структуры, каждая из которых содержит атрибуты всех пикселов экрана. Такие области называются страницами. В данный момент времени любая из страниц может отображаться видеоадаптером на дисплее, занимая при этом весь экран. Наличие страниц позволяет мгновенно менять изображение на экране, просто переключаясь с одной страницы на другую. Это

существенно улучшает "наглядность" программ, так как дает возможность проводить всю "черновую работу" по подготовке графического изображения на неотражаемой в настоящий момент странице.

Поскольку объем страницы видеопамяти ограничен, то количество битов, приходящееся на один пиксел, находится в обратной зависимости от общего количества пикселов на экране. Обычно атрибут пиксела состоит из 1, 2, 4 или 8 бит, в зависимости от графического режима. Все пикселы, имеющие одинаковое значение атрибута, отображаются на экране монитора одинаковым образом.

Если атрибуту каждого пиксела в видеопамяти отводится только один бит, то графика будет двухцветной, например черно-белой. Если каждый пиксел представляется N битами, то мы имеем возможность представить на экране одновременно 2^N оттенков; например, если на один пиксел отводится 4 бита, то получаем палитру из 16 цветов.

Далее, в системе программирования Turbo C все необходимые определения для графического модуля даны в файле graphics.h, и при выборе соответствующего режима он должен быть включен в программу (#include <graphics.h>). Стандартные функции для графической системы помещены в библиотеку graphics.lib, которая должна быть подключена в режиме Options.

За начало координат экрана (x=0, y=0) принимается его левый верхний угол. Значение x определяет число пикселов, на которое смещается вправо от начала координат указатель текущей позиции экрана – курсор. Значение y определяет аналогичное смещение вниз. При установке на экране графического окна за начало принимается левый верхний угол окна, который может быть смещен относительно левого верхнего угла экрана.

2. Инициализация графического режима

Если вы присоединили файл <graphics.h> , то можно инициализировать графический режим. Проще всего это сделать при помощи функции **initgraph**. Выглядит это следующим образом:

```
int gm, gd= DETECT;
initgraph (&gd,&gm," ");
```

Здесь gd и gm - переменные, которые определяют драйвер и режим его работы, соответственно.

Если значение gd не равно 0 (или DETECT), оно используется в качестве номера драйвера, который загружается в память, и система переводится в графический режим, соответствующий параметру gm. Если перед использованием функции initgraph переменной gd присвоить значение 0 (или DETECT), определение требуемого драйвера производится автоматически. Третий параметр этой функции содержит маршрут для поиска графического драйвера. Если в этом поле содержится пустая строка (" "), файл с драйвером должен находится на текущем диске. При инициализации очищается экран и устанавливаются по умолчанию всевозможные параметры системы, влияющие на работу графической функции, хотя в процессе работы эти параметры можно будет изменить из прикладной программы.

Обратная функция к initgraph

closegraph (void) – закрывает графический режим.

ЗАМЕЧАНИЕ. Здесь и далее void обозначает отсутствие параметров.

РЕКОМЕНДАЦИЯ! Чтобы быть уверенным, что с инициализацией графического режима у вас нет проблем, используйте функцию **graphresult** (void). Она .возвращает значение внутреннего кода ошибки и, если же он равен 0 (или grOk) , то можете смело работать дальше.

Подводя итоги, можно сказать, что грамотная инициализация графического режима выглядит следующим образом:

```
int gm, gd= DETECT;
initgraph (&gd,&gm," ");
if ( graphresult ( ) != grOk )
  {puts (" Ошибка в инициализации графического режима ");
   closegraph ( ) ;
  }
```

3. Основные графические примитивы

Ниже приводятся графические функции Turbo C для получения изображения на экране дисплея. Если дополнительно не указан тип параметров, переменные должны иметь целые значения. Угловые величины задаются в градусах и отсчитываются против направления часовой стрелки от горизонтальной линии, проведенной через центр.

Узнать размер экрана можно при помощи следующих функций:

getmaxx (void) – возвращает размер экрана в пикселах по горизонтали;

getmaxy (void) – возвращает размер экрана в пикселах по вертикали;

putpixel (x, y, цвет) - отображает точку с заданным цветом по координатам x, y;

moverel (x приращения, y приращения) – невидимо перемещает курсор в новую точку по заданным приращениям;

moveto (x, y) – невидимо перемещает курсор в новую точку с заданными координатами x, y;

line (x начала, y начала, x конца, y конца) – вычерчивает линию;

linerel (x приращения, y приращения) – вычерчивает линию из текущей точки по заданным приращениям;

lineto (x, y) – вычерчивает линию из текущей точки в точку с заданными координатами x, y;

rectangle (x левой стороны, y верхней стороны, x правой стороны, y нижней стороны) – вычерчивает прямоугольник;

drawpoly (число узловых точек, int m[]) – вычерчивает ломаную линию, соединяя отрезками прямых последовательность точек на плоскости; здесь m – указатель на массив записей из двух целых чисел – координат x и y узловых точек. В частности, чтобы нарисовать многоугольник, нужно задать одинаковые координаты для первой и последней точек.

Например, массив m для треугольника объявляется так:

```
struct xy   { int x, y;}  m[4];
```

Далее задаются координаты четырех точек:

```
m[0].x=10;        m[0].y=10;
m[1].x=50;        m[0].y=50;
m[2].x=50;        m[0].y=10;
m[3].x=m[0].x;    m[0].y= m[0].y;
```

Обращение к функции drawpoly для вычерчивания треугольника будет таким: drawpoly (4,(int *)m).

Массив m можно задать иначе:

```
int m[8];
```

Далее задаются координаты четырех точек:

```
m[0]=10;        m[1]=10;
m[2]=50;        m[3]=50;
m[4]=50;        m[5]=10;
m[6]=m[0];      m[7]= m[0];
```

В этом случае обращение к функции drawpoly для вычерчивания треугольника будет таким: drawpoly (4,m).

circle (x центра, y центра, радиус) – вычерчивает окружность;

arc (x центра, y центра, угол начала, угол конца, радиус) – вычерчивает дугу окружности;

ellipse (x центра, y центра, угол начала, угол конца, горизонтальный радиус, вертикальнй радиус) – вычерчивает дугу эллипса или эллипс, если разница между двумя заданными углами не меньше 360°;

Остановимся подробней на особенностях рисования дуг окружностей и эллипсов функциями arc и ellipse. Угол отсчитывается от угла, заданного аргументом "угол начала", к углу, заданному аргументом "угол конца". Переданные функциям значения углов "нормализуются", то есть заменяются эквивалентными значениями из интервала от 0 до 360 градусов. Таким образом, функции arc (x, y, -45, 45, radius), arc (x, y, 315, 45, radius) и arc (x, y, 675, -315, radius) рисуют одну и ту же дугу в четверть окружности.

Функции графического вывода изменяют координаты x, y курсора. Для их определения можно воспользоваться следующими функциями:

getx (void) – возвращает значение координаты x (в пикселах) для указателя текущей позиции;

gety (void) - возвращает значение координаты y (в пикселах) для указателя текущей позиции.

ЗАМЕЧАНИЕ. После использования функций circle, arc и ellipse следует аккуратнее работать с графическими примитивами, использующими текущую графическую позицию, так как значение координат (x, y) указателя текущей позиции, возвращаемое функциями getx и gety, может не соответствовать реальному местонахождению курсора. Поэтому, во избежание ошибок, рекомендуется устанавливать указатель текущей позиции в нужное место самостоятельно.

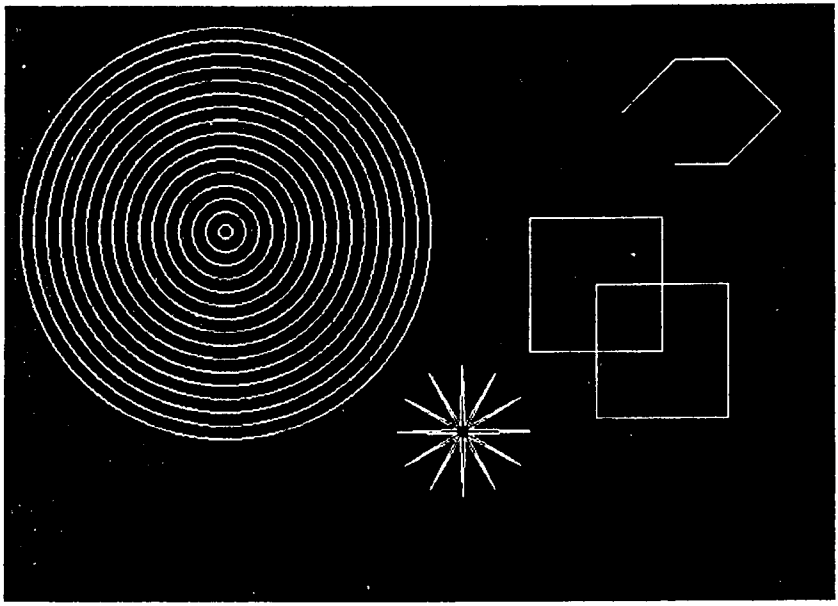
Далее приведем программы, использующие некоторые из вышеприведенных функций.

Программа рисует звезды и "разбегающиеся" окружности.

```
#include <graphics.h>
#include <math.h>
#include <stdio.h>
main ( )
{int h, x, y, x1, y1, x0, y0, i, radius, r, R;
 float a,k;                          // величины углов
//////////////////// инициализация графического режима ////////////////////////
 int gm, gd=DETECT;
 initgraph (&gd, &gm, "");
//////////////////////////////// рисование 16 окружностей ////////////////////////////
 for (i=0, radius=5; i<16; i++, radius+=10)
  circle (170, 170, radius);       // вывод окружности
/////////////////////////// рисование n-конечной звезды ////////////////////////////
```

```
// вычисления проводятся по формулам
//    x=R*cos (a); y=R*sin (a) и
//    x1=r*cos (a); y1=r*sin (a)
// где a=360 / (2*n) – величина изменения угла и n – число вершин
////////////////////////// рисование 12-конечной звезды ////////////////////////////////
// в этом случае n=12, следовательно, величина изменения угла
// a=360 / 24=15 градусов
// перевод градусов в радианы
 k=15*M_PI / 180;
// всего надо нарисовать 2*n=24 линии
 x0=350, y0=320; r=5; R=50;   // задание центра звезды
 x=R+x0; y=y0;                // задание начальной точки
 for (a=k, h=0, i=0; i<10; a+=k, i++)
  {// одна линия проводится от точки на описанной окружности до
   // точки на вписанной окружности, следующая линия - наоборот
   if (h)
  // вычисление экранных координат точки на описанной окружности
    {x1=R*cos (a)+x0; y1=R*sin (a)+y0; h=0;}
   else
 // вычисление экранных координат точки на вписанной окружности
    {x1=r*cos (a)+x0; y1=r*sin (a)+y0; h=1;}
     line (x, y, x1, y1);          // рисование линии
     x=x1, y=y1;                   // переопределение координат точки }
////////////////////////////////////  рисование ломаной  ////////////////////////////////////
 int  m [12];                      // определение массива вершин
 m[0]=470;    m[1]=80;             // координаты первой точки
 m[2]=510;    m[3]=40;
 m[4]=550;    m[5]=40;
 m[6]=590;    m[7]=80;
 m[8]=550;    m[9]=120;
 m[10]=510;   m[11]=120;
 drawpoly (6, m);
////////////////////////////  рисование двух прямоугольников  ////////////////////////
 rectangle (400, 160, 500, 260);
 rectangle (450, 210, 550, 310);
getchar ( );                       // задержка экрана
closegraph ( );                    // закрытие графического режима
}
```

Результат работы этой программы выглядит следующим образом



4. Вывод графических текстов

Turbo C представляет пользователю широкие возможности по выводу текстовой информации в графических режимах. Во-первых, это все функции стандартного вывода. Но все же эти функции ограничены стандартным видом и размером символов шрифта, а также возможностью размещения символов только в тех позициях экрана, которые допускаются в текстовом режиме. Библиотека графики Turbo C позволяет выводить на экран текст различными шрифтами. Turbo C имеет два типа шрифтов: битовый и сегментированный.

Каждому символу *битового шрифта* (bit-mapped font) ставится в соответствие матрица пикселов фиксированного размера. Turbo C использует в качестве битового шрифта таблицу знакогенератора для символов размером 8x8, установленную в компьютере перед инициализацией системы графики Turbo C. Все изменения таблицы знакогенератора, сделанные, например, программами русификации, будут сохранены. Это позволяет, применяя функции Turbo C, выводить текст русскими буквами и в графических режимах.

Другой тип шрифтов, используемых в Turbo C при выводе текста на экран, фактически задает правило "рисования" каждого символа. Он описывается как совокупность отрезков прямых линий, или сегментов. Этим и объясняется название шрифта *сегментированный* (stroke font). Программа может задать масштаб для каждого символа, "растягивая" или "сжимая" его по высоте либо ширине. Однако использование сегментированного шрифта для вывода текста несколько замедляет работу видеосистемы.

Файлы сегментированных шрифтов поставляются фирмой Borland International в закодированном виде в специальных файлах шрифтов, имеющих расширение .CHR и, подобно .BGI-драйверам, загружаются как оверлеи во время выполнения программы. На данный момент известно 10 сегментированных шрифтов, имена их файлов, количество и размеры символов приведены в таблице. Однако фирмой ведется разработка новых шрифтов и совершенствование существующих, поэтому значения в таблице могут не соответствовать имеющимся в вашем распоряжении версиям. В Turbo C доступны четыре сегментированных шрифта: Triplex, Small, Sans-Serif и Gothic. Для этих шрифтов в скобках приведены значения, относящиеся к CHR-файлам, входящим в дистрибутивный пакет Turbo C версии 2.0.

Далеко не все сегментированные шрифты имеют полный набор символов, т.е. пригодны для отображения всех 256 символов. Общими для всех шрифтов являются символы с кодами от 32 до 126 (цифры, латинские буквы и некоторые специальные знаки).

Номер шрифта	Имя файла	Количество символов	Вертикальный размер (в пикселах)
0	Встроен	256	8
1	TRIP.CHR	223 (96)	18
2	LITT.CHR	223 (96)	5
3	SANS.CHR	254 (126)	19
4	GOTH.CHR	223 (106)	19
5	SCRI.CHR	223	22
6	SIMP.CHR	223	21
7	TSCR.CHR	223	18
8	LCOM.CHR	223	21
9	EURO.CHR	223	33
10	BOLD.CHR	248	36

Библиотека графики Turbo C дает возможность выводить текст в графических режимах слева направо и снизу вверх. Для вывода символов при использовании любого шрифта может быть задан масштаб знакоместа по отношению к знакоместу 8x8. При использовании сегментированного шрифта может быть задан размер знакоместа и в относительных единицах масштаба как по вертикали, так и по горизонтали. Кроме того, выводимые строки текста могут выравниваться по-разному: строка может быть "прижата" влево и вверх относительно точки, определенной как текущая точка отсчета, влево и вниз и т.п.

Установить стиль текста можно при помощи функции

settextstyle (шрифт, направление, размер символа) – выбирает шрифт, устанавливает направление и размер знакоместа для последующего вывода текстовой информации.

Параметр "шрифт" может принимать значения от 0 до 10, из них только первое не требует загрузки новых генераторов знаков. Если задан недопустимый номер шрифта (то есть положительный и не совпадающий с номером какого-либо из векторных шрифтов, установленных в данный момент в графической системе), то инициализируется встроенный матричный шрифт, но одновременно устанавливается код ошибки – 14.

Второй параметр имеет значения:

0 (HORIZ_DIR)– для горизонтального слева направо направления текста,

1 (VERT_DIR)– для вертикального сверху вниз направления текста; в этом случае символы будут повернуты на 90 градусов против хода часовой стрелки. Кроме того, использование вертикального расположения строки приводит к изменению пропорций выводимых символов – они становятся ниже и шире, чем в горизонтальной строке. Если было задано значение аргумента "направление" большее единицы, то устанавливается режим вывода с вертикальным расположением строк. Отрицательное значение этого аргумента приводит к аварийному завершению всей программы.

Размер символа – это цифра, показывающая, во сколько раз увеличивается его величина. Если, например, задать "размер символа" равным 5,то символ будет изображаться в знакоместе 40x40. Любое значение аргумента "размер символа", выходящее за пределы диапазона от 0 до 10, при выполнении функции заменяется на 10. Если эта цифра 0, то для битового шрифта размер знакоместа изменяться не будет. Для сегментированного шрифта размер знакоместа задается пользователем с помощью функции

setusercharsize (multx, divx, multy, divy) – пользователь устанавливает размер символа (изменение по горизонтали в multx/divx и по вертикали в multy/divy раз); масштаб задается относительно знакоместа 8x8. В случае изменения размера параметр "шрифт" в предыдущей функции должен принимать значение 0. Например,

setusercharsize (4, 1, 3, 2);

устанавливает знакоместо в 4 раза шире и 3/2 раз выше, чем знакоместо 8x8, то есть 32x12 пикселов. Функция предполагает использование сегментированных шрифтов. Для битового шрифта масштаб по вертикали и горизонтали всегда одинаков и задается функцией settextstyle. Предварительно функция проверяет значения аргументов. Если хотя бы один из них отрицательный, то функция завершает работу, на изменяя при этом текущих значений масштаба. Если любой из аргументов divx или divy равен 0, то он автоматически заменяется на 1, и функция выполняется до конца.

ЗАМЕЧАНИЕ. Если функция setusercharsize вызывается после функции settextstyle, режим независимого масштабирования по осям устанавливается ею при любом значении аргумента "размер символа". Если же векторный шрифт инициализировался функцией settextstyle с нулевым значением аргумента "размер символа", то этот режим может быть установлен функцией setusercharsize, выполненной даже до вызова функции settextstyle. Во всех случаях функция setusercharsize устанавливает новые размеры символов, модифицируя по описанным выше правилам размеры их базового варианта.

При инициализации графического режима устанавливается такой режим работы знакогенератора, как если бы была выполнена функция

settextstyle (DEFAULT_FONT, HORIZ_DIR, 1);

то есть будет использоваться встроенный матричный шрифт с размером знакоместа 8x8 пикселов, и символы будут выводиться горизонтальными строками.

Графическая система Turbo C позволяет позиционировать на странице видеопамяти выводимый графический текст с точностью до пиксела. Это выгодно ее отличает от стандартных средств вывода, всегда привязанных к фиксированной сетке знакомест символов текстового режима. Поскольку функции вывода графических текстов устроены так, что за один раз выводят только одну строку, то им для позиционирования текста достаточно указать расположение выводимой строки относительно некоторой опорной точки страницы. В качестве этой точки может быть выбрана текущая графическая позиция, либо произвольная точка страницы видеопамяти.

Расположением выводимой строки относительно опорной точки управляет функция

settextjustify (по горизонтали, по вертикали) – устанавливает выравнивание текста; каждый параметр может принимать значения от 0 до 2.

По горизонтали:

LEFT_TEXT (0) – левая граница строки "прижимается" справа к вертикальной линии, проведенной через опорную точку;

CENTER_TEXT (1) – вертикальная линия, проведенная через опорную точку, проходит через середину строки;

RIGHT_TEXT (2) – правая граница строки "прижимается" слева к вертикальной линии, проведенной через опорную точку;

По вертикали:

BOTTOM_TEXT (0) – нижняя граница строки "прижимается" сверху к горизонтальной линии, проведенной через опорную точку;

CENTER_TEXT (1) – горизонтальная линия, проведенная через опорную точку, проходит через середину строки;

TOP_TEXT (2) – верхняя граница строки "прижимается" снизу к горизонтальной линии, проведенной через опорную точку;

Отметим, что при выводе текста вертикально установки выравнивания по горизонтали LEFT_TEXT и RIGHT_TEXT не различаются и аналогичны RIGHT_TEXT.

Координаты опорной точки задаются либо явно функцией outtextxy(), либо используются текущие значения.

Функция settextjustify хорошо защищена от запрещенных значений аргументов. В этих случаях она устанавливает код ошибки – 11 и сохраняет предыдущий режим позиционирования неизмененным.

При инициализации графического режима устанавливается следующий режим позиционирования

settextjustify (LEFT_TEXT, TOP_TEXT);

В графической библиотеке Turbo C существует функция, позволяющая получить информацию о текущем режиме вывода графических текстовых сообщений:

gettextsetting (struct textsettingstype far *texttypeinfo) – возвращает значения параметров текста, являющихся элементами записи texttypeinfo типа

```
struct textsettingstype
  {int font;
   int direction;
   int charsize;
```

```
    int horiz;
    int vert;
  }.
```

В графической библиотеке предусмотрены еще две справочные функции, полезные при работе с графическими текстами. Для правильного размещения текста на странице бывает необходимым знать размеры занимаемого им пространства. Это позволяют сделать следующие функции:

textheight (char far * textstring) – возвращает значение высоты символа;

textwindht (char far * textstring) – возвращает значение ширины символа.

Они возвращают высоту и ширину (в пикселах) прямоугольника, в который был вписан текст при выводе строки textstring знакогенератором текущего графического шрифта с учетом установленного в данный момент масштаба. Сам текст при этом не выводится. Хотя значения, возвращаемые функциями textheight и textwindht, не зависят от расположения текста на странице видеопамяти, но из-за "неквадратности" пиксела размеры одной и той же строки, выведенной горизонтально и вертикально, могут различаться.

Наконец, перейдем к описанию функций, непосредственно осуществляющих вывод графических текстов на активную страницу видеопамяти. При работе они последовательно создают в видеопамяти рисунок символов переданной им строки, используя для этого встроенный в Turbo C матричный (для битового шрифта) или векторный (для сегментированного шрифта) знакогенератор и таблицу символов активного в данный момент графического шрифта. Цвет символов устанавливается функцией setcolor. Никаких битовых операций с атрибутами пикселов видеопамяти не предусмотрено. Линии, которыми рисуются символы сегментированных шрифтов, всегда сплошные тонкие; они не могут быть изменены при помощи функции setlinestyle.

Вывод графического текста можно осуществить, используя следующие функции:

outtext (char far * textstring) – выводит на экран строку, на которую указывает textstring; здесь в качестве опорной точки принимается текущая графическая позиция;

outtextxy (x, y, char far * textstring) – выводит на экран строку, на которую указывает textstring; здесь в качестве опорной точки принимается точка с координатами (x, y), координаты задаются в системе координат текущего графического окна.

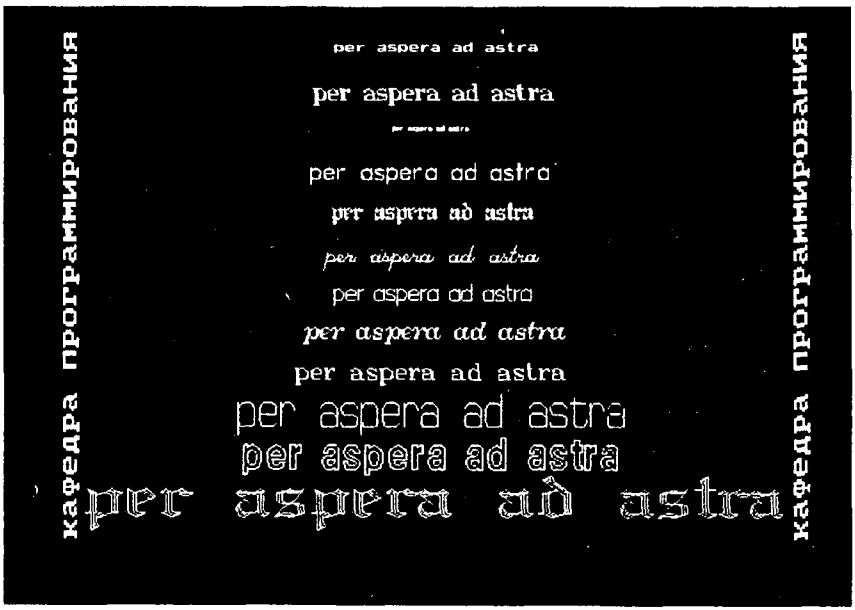
Если текст выводится в графическое окно с включенным усечением, он усекается на границах окна. Для сегментированных шрифтов усечение производится с точностью до пикселов, для битовых шрифтов оно происходит с точностью до символа. Усечение строки может выполняться по одной границе или по обеим границам сразу, если задается выравнивание по центру. Если среди выводимых символов строки содержится символ, не включенный в сегментированный шрифт, функция пропускает его, не изменяя текущих позиций по горизонтали или вертикали.

В качестве примера приведем программу, выводящую текст "per aspera ad astra" ("сквозь тернии к звездам") и "кафедра программирования" всевозможными шрифтами:

```
#include <graphics.h>
#include <stdio.h>
main ( )
{int y, i;
 char *s="кафедра программирования",
      *s1="per aspera ad astra";
////////////////////// инициализация графического режима //////////////////////////
 int gm, gd=DETECT;
 initgraph (&gd, &gm, "");
// установка выравнивания текста
// здесь выравнивание по горизонтали и
// по вертикали производится по центру
 settextjustify(1,1);
// текст выводится по горизонтали 11 шрифтами
 for (i=0, y=20; i<=10; i++, y+=35)
   {// установка стиля текста
    // размер шрифта стандартный
    settextstyle (i, 0, 1);
    // вывод текста по горизонтали
    // опорная линия – вертикальная линия в центре экрана
    outtextxy (getmaxx ( ) / 2, y, s1);
   }
 settextstyle (4, 0, 0); // установка готического стиля текста
// установка нового размера символа:
// по горизонтали шире в 2 раза, по вертикали выше в 3/2 раза
 setusercharsize (2, 1, 3, 2);
// вывод текста по горизонтали
 outtextxy (getmaxx ( ) / 2, y, s1);
```

```
// установка стиля текста
// вывод по вертикали, символы в 2 раза больше
 settextstyle (0, 1, 2);
// вывод текста
// опорная линия – горизонтальная линия в центре экрана
 outtextxy (25, getmaxy ( ) / 2, s);
 outtextxy (615, getmaxy ( ) / 2, s);
 getchar ( );                          // задержка экрана
// закрытие графического режима
 closegraph ( );
}
```

Результат работы программы имеет следующий вид:



Далее приведем текст программы построения графика функции.

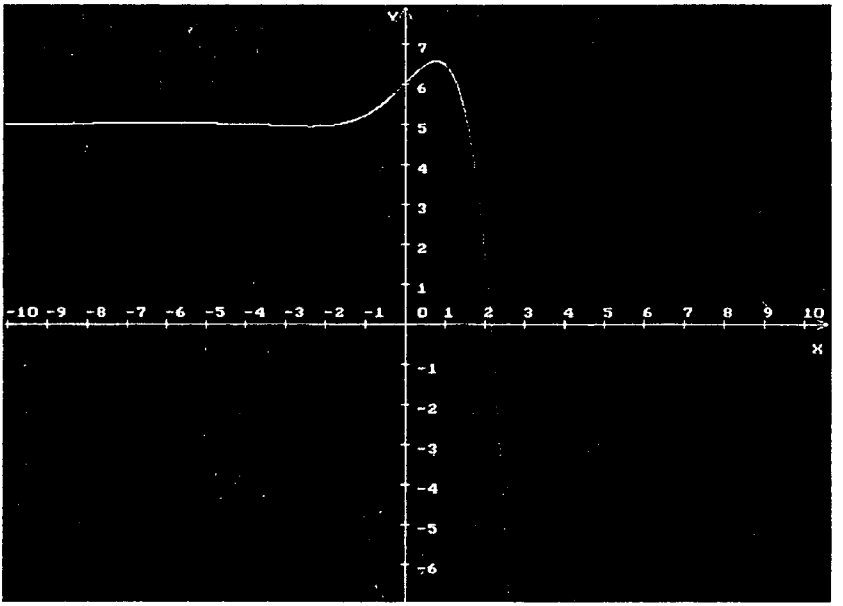
```
#include <graphics.h>
#include <math.h>
#include <stdio.h>
```

```
#include <stdlib.h>
// начало рассматриваемого интервала
#define Nst  -15
// конец рассматриваемого интервала
#define Nend 15
#define STEP 0.01 // шаг
// единичный отрезок по оси Ox
#define EDX  30
// единичный отрезок по оси Oy
#define EDY  30
// исследуемая функция
#define F (x)  cos (x) * exp (x) + 5
main ( )
{int maxx, maxy, i;
 float x, y, x0, y0, xx, yy;
 char *s;
 //////////////////// инициализация графического режима ////////////////////////////
 int gm, gd=DETECT;
 initgraph (&gd, &gm,"");
// определение максимального размера экрана
 maxx=getmaxx (); maxy=getmaxy ();
// нахождение середины экрана
 x0=maxx/2; y0=maxy/2;
 line (x0, maxy, x0, 1);              // провели линию – ось Oy
// стрелки – направление оси
 line(x0-5, 10, x0, 1); line(x0+5, 10, x0, 1);
 outtextxy (x0-13, 5, "Y");          // вывели название оси
// координаты разделяющей черты
 x=x0-3; xx=x0+3;
////////////////// нанесение координатной сетки на ось Oy ////////////////////////
 for (y=y0+EDY, yy=y0-EDY, i=1; y<maxy; y+=EDY, yy-=EDY, i++)
  {// вывод разделяющей черты и соответствующего ей номера в
   // положительном направлении оси Oy
   line (x, yy, xx, yy);
   // перевод целого числа в строку
   itoa (i, s, 10);
   outtextxy (x0+10, yy, s);          // вывод этой строки
   // аналогичные действия для отрицательной полуоси
   line (x, y, xx, y);
   itoa (-i, s, 10);
```

```
  outtextxy (x0+10, y, s);
 }
 line (1, y0, maxx-1, y0);          // провели линию – ось Ox
// стрелки – направление оси
 line (maxx-10, y0-3, maxx-1, y0);
 line (maxx-10, y0-3, maxx-1, y0);
 outtextxy (maxx-13, y0+15, "X");// вывели название оси
// координаты разделяющей черты
 y=y0-3; yy=y0+3;
////////////////////  нанесение координатной сетки на ось Ox  ////////////////////
 for (x=x0+EDX, xx=x0-EDX, i=1; x<maxx; x+=EDX, xx-=EDX, i++)
  {// вывод разделяющей черты и соответствующего ей номера в
   // положительном направлении оси Ox
   line (x, y, x, yy);
   // перевод целого числа в строку
   itoa (i, s, 10);
   outtextxy (x, y0-12, s);            // вывод этой строки
 // аналогичные действия для отрицательной полуоси
   line (xx, y, xx, yy);
   itoa (-i, s, 10);
   outtextxy (xx, y0-12, s);
  }
// вывод цифры для начала координат
 outtextxy (x0+10, y0-12, "0");
//////////  вычисление значений заданной функции на данном  ////////////////
////////////////////  интервале и вывод этой точки на экран  ////////////////////
////////////////  от начала до конца интервала с шагом STEP  ////////////////////
 for (x=Nst; x<=Nend; x+=STEP)
  {// вычислили значение исследуемой функции
   y=F (x);
   // пересчет полученных координат (x, y)
   //:в экранную систему координат: (xx, yy)
   xx=x0+EDX*x;
   yy=y0-EDY*y;
   // вывод на экран точки с координатами (xx, yy) желтым цветом
   putpixel ((int) xx, (int) yy, 14);
  }
 getchar ( );                        // задержка экрана
// закрытие графического режима
 closegraph ( );
```

}

Результат работы программы имеет следующий вид:



5. Параметры и атрибуты графического вывода

Многие из основных графических примитивов могут быть "залиты" текущим цветом с использованием текущего шаблона или маски заполнения. После инициации графической системы и установки нужного видеорежима возможен выбор необходимых цветов пикселов.

Возможности по выбору цветов принципиально различны для различных адаптеров. Число цветов в палитре можно выяснить при помощи функции getmaxcolor.

getmaxcolor (void) – возвращает максимальное число цветов, которое может отображаться на экране. В зависимости от режима, в котором проведена инициализация графической системы, возвращаемое значение может быть равно 1, 3 или 15 (следует учесть, что счет начинается с 0).

Используемые номера цветов:

Символическая константа	Значение	Цвет на экране монитора
BLACK	0	Черный
BLUE	1	Синий
GREEN	2	Зеленый
CYAN	3	Голубой
RED	4	Красный
MAGENTA	5	Фиолетовый
BROWN	6	Коричневый
LIGHTGRAY	7	Светло-серый
DARK GRAY	8	Темно-серый
LIGHTBLUE	9	Светло-синий
LIGHTGREEN	10	Светло-зеленый
LIGHTCYAN	11	Светло-голубой
LIGHTRED	12	Светло-красный
LIGHTMAGENTA	13	Светло-фиолетовый
YELLOW	14	Желтый
WHITE	15	Белый

setbkcolor (номер цвета) – устанавливает цвет фона.

ЗАМЕЧАНИЕ. Если цветом фона был выбран любой цвет из палитры, кроме черного, то черный цвет как бы "выпадает" из палитры, то есть при обращении к цвету со значением 0, мы получаем цвет фона. Можно либо заменить черный цвет на темно-серый, либо восстановить его при помощи функции setpallette (0, 0).

Обратная ей функция

getbkcolor (void) – позволяет установить код цвета, выбранный в качестве цвета фона в текущий момент;

setcolor (номер цвета) – устанавливает цвет графических образов. До того момента, пока цвет не установлен, используется максимальный (из палитры) номер цвета. В случае, если задан недопустимый номер цвета, текущий цвет остается неизменным;

getcolor (void) – возвращает номер текущего цвета линий;

floodfill (x,y, номер цвета границы) – заполняет текущим стилем область экрана, ограниченную непрерывной линией с указанным номером цвета границы; заполнение начинается с точки с координатами (x, y). Функция заполняет область либо внутри замкнутой линии, либо вне ее. Это зависит от положения начальной точки:

если точка лежит внутри области, заполняется внутренняя область;

если точка лежит вне замкнутой области, заполняется внешняя область;

если точка лежит точно на линии указанного цвета, заполнение не производится.

Заполнение начинается с начальной точки и продолжается во всех направлениях, пока не встретится пиксел с цветом границы. Цвет границы должен отличаться от цвета заполнения, в противном случае будет заполнен весь экран. Если заполняемая область ограничена не сплошной линией, заполнение "протекает" через отверстия. Цвет и маска заполнения могут быть заданы при помощи функций setfillstyle и setfillpattern.

Для удобства пользователей библиотека графики Turbo C содержит целую группу предопределенных комбинаций (символ/цвет) для заполнения областей экрана. Пару значений символов/цвета часто называют *стилем заполнения* (filling style). Другими словами, стиль заполнения – это маска заполнения + цвет заполнения.

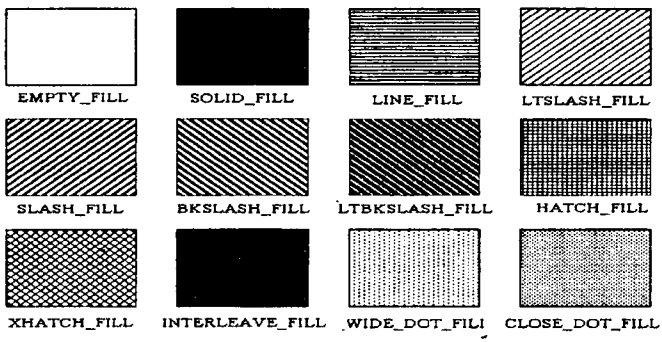
Маска заполнения позволяет задать способ заполнения отдельных областей экрана. Она определяется восьмибайтовм шаблоном, рассматриваемым как битовая карта 8x8. Заполняемая область также разбивается на блоки (знакоместа) по 8x8 пикселов, Маска "накладывается" на каждое такое знакоместо по следующему правилу: если соответствующий бит в маске заполнения равен 1, то пиксел в знакоместе имеет номер текущего цвета; в противном случае пиксел остается неизменным.

setfillstyle (номер наполнителя, номер цвета) – устанавливает стиль наполнителя. Если значение аргумента "номер цвета" выйдет за пределы возможной палитры, то установится максимальное значение цвета. Номер наполнителя может изменяться от 0 до 12. Последняя цифра (12) указывает, что маска заполнения будет задана пользователем.

Имеются следующие предопределенные маски:

Символическая константа	Значение	Описание стиля заполнения
EMTY_FILL	0	Заполнение цветом фона
SOLID_FILL	1	Заполнение текущим цветом
LINE_FILL	2	Заполнение символами ---
LTSLASH_FILL	3	Заполнение символами /// нормальной толщины
SLASH_FILL	4	Заполнение символами ///

		удвоенной толщины
BKSLASH_FILL	5	Заполнение символами \\\ удвоенной толщины
LTBKSLASH_FIL L	6	Заполнение символами \\\ нормальной толщины
HATCH_FILL	7	Заполнение вертикально-горизонтальной штриховкой тонкими линиями
XHATCH_FILL	8	Заполнение штриховкой крест-накрест по диагонали "редкими" тонкими линиями
INTERLEAVE_FI LL	9	Заполнение штриховкой крест-накрест по диагонали "частыми" тонкими линиями
WIDE_DOT_FILL	10	Заполнение "редкими" точками
CLOSE_DOT_FIL L	11	Заполнение "частыми" точками
USER_FILL	12	Заполнение по определенной пользователем маске заполнения



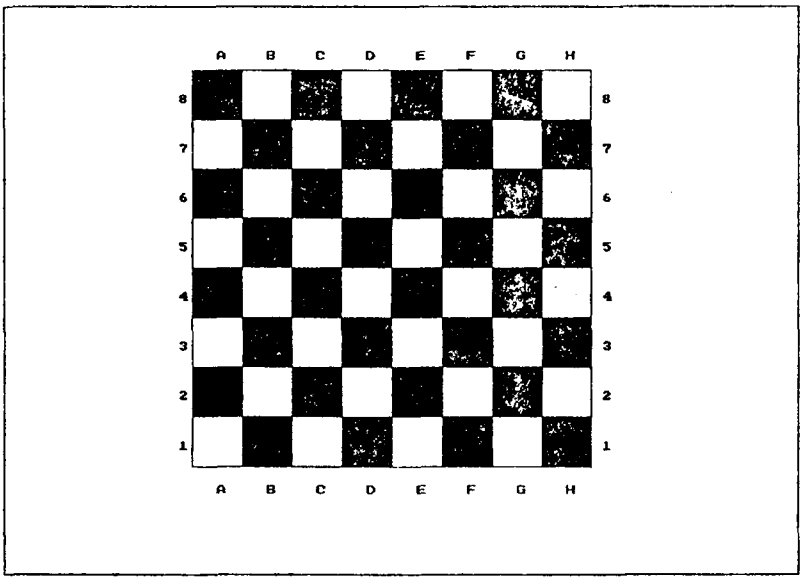
ЗАМЕЧАНИЕ. Следует учесть, что установка цвета фона (при помощи фунции setbkcolor) и заливка всего экрана этим же цветом (функция floodfill) дают визуально одинаковый результат, но с разными качествами. Так, например, линия того же цвета в первом случае воспринимается как граница для последующих заполнений, а во втором случае – нет.

В качестве иллюстрации вышеприведенных функций приведем следующую программу, рисующую шахматную доску:

```
#include <graphics.h>
#include <stdio.h>
//////////////////////////  определение координат доски  ////////////////////////////////
// верхней левой точки
#define  x1   160
#define  y1   50
// правой нижней точки
#define  x2   480
#define  y2   370
// длина одного квадрата
#define  kx    (x2-x1) / 8
// ширина одного квадрата
#define  ky    (y2-y1) / 8
main ( )
{char *s1 [8]={"A", "B", "C", "D", "E", "F", "G", "H"},
      *s2 [8]={"1", "2", "3", "4", "5", "6", "7", "8"};
 int x, y, i;
 ////////////////////  инициализация графического режима  ////////////////////////////
 int gm, gd=DETECT;
 initgraph (&gd, &gm, "");
 setbkcolor (15);                    // установка цвета фона (белый)
 setcolor (8);                       // установка цвета линий (темно-серый)
 ////////////////////////////////////  рисование доски  ////////////////////////////////////
 // установка стиля (сплошной) и цвета (белый) заливки
 setfillstyle (SOLID_FILL, 15);
 rectangle (x1, y1, x2, y2);         // нарисовали квадрат
 // залили его установленным стилем и цветом
 floodfill (200, 200, 8);
 // вертикальными линиями разбили квадрат на 8 прямоугольников
 for (x=x1+kx; x<x2; x=x+kx)     line (x, y1, x, y2);
 // горизонтальными линиями разбили
 // каждый прямоугольник на 8 квадратов
 for (y=y1+ky; y<y2; y=y+ky)     line (x1, y, x2, y);
 //////////////////////////  вывод букв и цифр на доске  ////////////////////////////////
 for (x=x1+kx / 2, i=0; x<x2; x=x+kx, i++)
   {outtextxy (x, y2+15, s1[i]);     // вывод букв по горизонтали
    outtextxy (x, y1-15, s1[i]);
   }
```

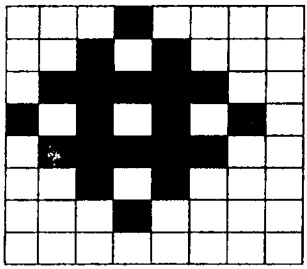
```
 for (y=y1+ky / 2, i=7; y<y2; y=y+ky, i--)
   {outtextxy (x1-10, y, s2[i]);       // вывод цифр по вертикали
    outtextxy (x2+10, y, s2[i]);
   }
// установка стиля (сплошной) и цвета (темно-серый) заливки
 setfillstyle (SOLID_FILL, 8);
// залили через один квадрат установленным стилем и цветом
 for (x=x1+kx / 2; x<x2; x=x+2*kx)
   for (y=y1+ky / 2; y<y2; y=y+2*ky)        floodfill (x, y, 8);
// залили оставшиеся квадраты установленным стилем и цветом
 for (x=x1+3*kx / 2; x<x2; x=x+2*kx)
   for (y=y1+3*ky / 2; y<y2; y=y+2*ky)     floodfill (x, y, 8);
 getchar ( );                          // задержка экрана
// закрыли графический режим
 closegraph ( );
}
```

На экране результат работы этой программы выглядит следующим образом:



Далее,

setfillpattern (char far *up, номер цвета) – задает цвет и маску для заполнения областей экрана. По умолчанию используется белый цвет и маска заполнения, состоящая из матрицы единиц во всех битах. Таким образом, по умолчанию все пикелы заполняемой области имеют белый цвет. Аргумент up указывает на начало области из 8 байт, задающих новую маску заполнения. Предположим, мы хотим использовать следующую маску:



00010000	10
00101000	28
01111100	7C
10101010	AA
01111100	7C
00101000	28
00010000	10
00000000	00

т.е.8 шестнадцатеричных чисел.

Каждой строке матрицы соответствует двухразрядное шестнадцатеричное число. Его первая цифра представляет двоичное число первой половины матрицы, а вторая цифра – двоичное число второй половины. Например, 00010000 = 0x10. Здесь первые две цифры 0x (нуль икс) говорят о том, что число шестнадцатеричное. Таким образом, чтобы получить нашу маску заполнения следует объявить следующий массив p:

char p[ ]={0x10, 0x28, 0x7C, 0xAA, 0x7C, 0x28, 0x10, 0x00}.

Вызов функции setfillpattern для выбранного стиля заполнения и задание для него красного цвета будет осуществляться так:

setfillpattern (p, 4).

Задав стиль и маску заполнения, можно вывести некоторые графические примитивы, заполненные этим стилем.

bar (x левой стороны, y верхней стороны, x правой стороны, y нижней стороны) – вычерчивает закрашенный прямоугольник, не выделяя его внешнего контура;

bar3d (x левой стороны, y верхней стороны, x правой стороны, y нижней стороны, глубина, p) – вычерчивает прямоугольный параллелепипед с закрашенной лицевой гранью;

если p=1, верхняя грань присутствует;

если p=0, верхняя грань отсутствует (эта возможность бывает полезной, когда нужно поставить несколько параллелепипедов друг на друга);

если сделать аргумент "глубина" равным 0, то можно нарисовать обычный прямоугольник, обрамленный внешним контуром.

fillellipse (x центра, y центра, горизонтальный радиус, вертикальнй радиус) –вычерчивает закрашенный эллипс;

fillpoly (число узловых точек, int m[ ]) – вычерчивает закрашенный многоугольник (параметр m определяется также, как и для функции drawpoly).

ЗАМЕЧАНИЕ. Однако, если функция drawpoly допускает незамкнутые контура, то функция fillpoly всегда соединяет последнюю точку в полученном ею списке с первой, автоматически замыкая контур. И еще следует добавить, что с контурами без самопересечений эта функция справляется легко, но уже не всякий массив с повторяющимися точками будет проинтерпретирован правильно.

pieslice (x центра, y центра, угол начала, угол конца, радиус) - вычерчивает закрашенный сектор круга;

sector (x центра, y центра, угол начала, угол конца, горизонтальный радиус, вертикальнй радиус) – вычерчивает закрашенный сектор эллипса.

Особенностью функций pieslice и sector по сравнению с функциями arc и ellipse является то, что после нормализации значений аргументов "угол начала" и "угол конца", то есть приведения их к диапазону от 0 до 360 градусов, сектор всегда будет строится от меньшего значения угла к большему (а не от значения "угол начала" до значения "угол конца", как это было у функций arc и ellipse). Из-за этого невозможно заставить функции pieslice и sector изобразить сектор, пересекающий положительное направление оси X. Кроме того, при задании любого другого стиля линии, отличного от 0 (сплошная линия), дуга сектора становится невидимой. На стиль радиуса это не распространяется.

Далее, при выводе отрезков прямых линий система графики Turbo C позволяет определить такой параметр как *стиль линии.* Turbo C поддерживает ряд предопределенных стилей линий, но, как и в случае маски заполнения, можно описать собственный стиль линии. Выбор подходящего стиля выполняет функция

setlinestyle (стиль линии, образец битового отрезка, толщина) – устанавливает характер и толщину линий геометрических объектов. Толщина линий может быть равна 1 или 3 пикселам. Она задается либо целым числом, либо символической константой:

1 – NORM_WIDTH;

3 – THICK_WIDTH.

ЗАМЕЧАНИЕ. После выполнения любой из функций arc, circle или ellipse сбрасывается установка повышенной толщины линии, если таковая была сделана ранее.

Стиль линии задается либо целым числом, либо символической константой.

Символическая константа	Значение	Описание стиля линии
SOLID_LINE	0	Сплошная (непрерывная) линия
DOTTED_LINE	1	Линия из точек
CENTER_LINE	2	Штрих-пунктирная линия
DASHED_LINE	3	Штриховая линия
USERBIT_LINE	4	Определенная пользователем линия

Таким образом, если использовать предопределенный стиль линии (то есть 1-3), то второй параметр в функции setlinestyle должен быть равен 0. Если же стиль линии сделать равный 4, то пользователь сможет сам создать шаблон, с помощью которого можно создать любой периодически повторяющийся рисунок линии с периодом в 16 пикселов. Если некоторый бит шаблона равен 1, то соответствующий пиксел линии рисуется, в противном случае – нет. Так, сплошной линии соответствует шаблон 0xFFFF,а пунктир может задаваться, например, шаблоном 0x0F0F (0000111100001111). Самостоятельно определить стиль линии можно так:

setlinestyle (4, 0xAA, 3) (0000101000001010).

Определенный пользователем стиль линии действует при любой толщине линии .

Изначальная установка характеристик линий при инициализации графической системы соответствует значениям SOLID_LINE и NORM_WIDTH, то есть рисуются сплошные линии толщиной в один пиксел.

ЗАМЕЧАНИЕ. Если аргументу "толщина" присваивать недопустимые значения 0 или 2, то функция принимает значение 1 (NORM_WIDTH). Если же при вызове функции setlinestyle первые два аргумента принимают недопустимые значения, то функция игнорируется и прежние установки сохраняются.

Для определения текущей установки стиля линии используется функция

getlinesetting (struct linesettingstype far *lineinfo) – заносит по указанному адресу информацию о характеристиках линии в формате, задаваемом следующим описанием типа:

```
struct linesettingstype
  { int linestyle;        /* идентифицирует стиль линии */
    unsigned upattern; /* задает определенный пользователем стиль
                          линии; имеет значение только тогда, когда
                          linestyle = 4 */
    int thickness;        /* задает толщину линии */
  };
```

Поля этой структуры соответствуют аргументам функции setlinestyle.

При выводе отрезков прямых линий в графическом режиме система графики Turbo C позволяет дополнительно задать *режим вывода линии,* то есть имеется возможность указать способ, которым код рисующего цвета будет взаимодействовать с атрибутами пикселов, уже находящихся на странице видеопамяти на месте рисуемого объекта. Выбор одного из возможных способов осуществляется функцией

setwritemode (номер режима) – устанавливает режим вычерчивания линий. Аргумент может принимать только одно из двух значений 0 или 1 (любое другое значение аргумента перед выполнением функции автоматически берется по модулю 2):

0 (COPY_PUT) – новые линии "накладываются" на старое изображение;

1 (XOR_PUT) – между точками новой линии и имеющимся на экране изображением выполняется операция Исключающее Или (эта операция дает истинный результат, если по выбранным координатам точка есть хотя бы в одном изображении, но не в обоих вместе. Истинность результата приводит к появлению новой или сохранению старой точки на экране). В частности, можно стереть выведенную линию с экрана, выполнив вывод линии еще раз.

ЗАМЕЧАНИЕ. Однако следует сказать, что функция setwritemode воздействует в соответсвии с приведенным выше описанием и без побочных эффектов только на кусочно-линейные графические примитивы (например, на кривые второго порядка функция действует с побочными эффектами).

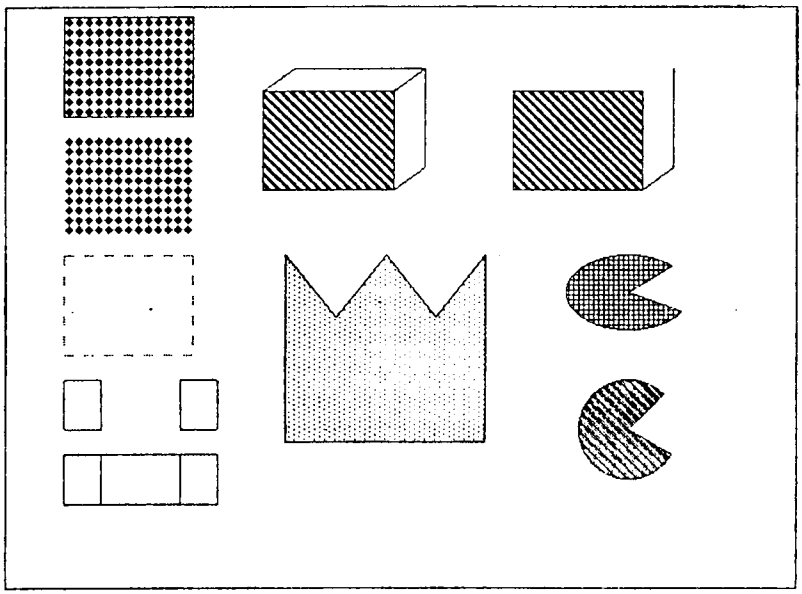
Далее рассмотрим программу, использующую вышеприведенные функции.

```
#include <graphics.h>
```

```
#include <stdio.h>
main ( )
{////////////////////// инициализация графического режима ////////////////////////
 int gm, gd=DETECT;
 initgraph (&gd, &gm, "");
 setbkcolor (15);                    // установка цвета фона (белый)
 setcolor (8);                       // установка цвета линий (темно-серый)
/////////// вывод на экран двух прямоугольников, закрашенных /////////////
///////////////// установленным пользователем стилем заливки /////////////////
// установка стиля заливки (задается пользователем)
 setfillstyle (12, 8);
// формирование стиля заливки
 char p[ ]={0x10, 0x28, 0x7c, 0xaa, 0x7c, 0x28, 0x10, 0x00};
 setfillpattern (p, 8);              // установка маски заливки
 rectangle (47, 7, 151, 87);         // вывод прямоугольника
 floodfill (50, 10, 8);              // залили прямоугольник
// нарисовали залитый тем же стилем прямоугольник
 bar (47, 103, 151, 183);
 setfillstyle (5, 8);                // сменили стиль заливки
////////////// вывод параллелепипеда во всевозможных режимах ///////////////
 bar3d (207, 67, 311, 147, 25, 1);
 bar3d (407, 67, 511, 147, 25, 0);
//////////////////////////// вывод залитого многоугольника ////////////////////////
// задание координат точек фигуры
 struct  figura  {int x, y;} m [8];
 m [0].x = 225;    m [0].y = 200;      // координаты первой точки
 m [1].x = 225;    m [1].y = 350;
 m [2].x = 385;    m [2].y = 350;
 m [3].x = 385;    m [3].y = 200;
 m [4].x = 345;    m [4].y = 250;
 m [5].x = 305;    m [5].y = 200;
 m [6].x = 265;    m [6].y = 250;
 m [7].x = 225;    m [7].y = 200;
 setfillstyle (10, 8);               // сменили стиль заливки
 fillpoly (7, (int *) m);            // вывели залитый этим стилем многогранник
 setlinestyle (4, 0xaa, 1);          // установили свой стиль линий
rectangle (47, 200, 151, 280); // нарисовали этим стилем прямоугольник
 setlinestyle (0, 0, 1);             // вернули сплошную тонкую линию
///////////////// демонстрация работы функции setwritemode ////////////////////
// установили режим вывода линий “копирование”
```

```
 setwritemode (1);
// вывели в этом режиме два пересекающихся прямоугольника
 rectangle (47, 300, 141, 340); rectangle (77, 300, 171, 340);
// сменили режим вывода линий на "Исключающее Или"
 setwritemode (0);
/ / вывели в этом режиме два пересекающихся прямоугольника
 rectangle (47, 360, 141, 400); rectangle (77, 360, 171, 400);
//////////////// демонстрация работы функций sector и pieslice ////////////////////
 setfillstyle (7, 8);                    // сменили стиль заливки
// вывели залитый этим стилем сектор эллипса
 sector (500, 230, 45, -45, 50, 30);
 setfillstyle (6, 8);                    // сменили стиль заливки
// вывели залитый этим стилем сектор круга
 pieslice (500, 340, 45, -45, 40);
 getchar ();                             // задержка экрана
closegraph ();                           // закрытие графического режима
}
```

Результат работы программы имеет вид:



6. Базовые функции доступа к видеопамяти

Как отмечалось выше, многие адаптеры позволяют хранить в видеопамяти сразу несколько страниц, которые нумеруются, начиная с 0. Страница, отображаемая в данный момент на экране, называется *видимой.* Страница памяти, на которую выполняется вывод графической информации, называется *активной.* Обычно активная страница является одновременно и видимой, поэтому любой вывод информации на экран приводит к изменению изображения на экране. Первоначально активной и видимой страницами устанавливается страница с номером 0, однако возможен и режим работы, в котором видимая и активная страницы не совпадают. В этом случае возможно формирование картинки "за кулисами", то есть вывод графической информации не будет изменять изображение на экране. Turbo C имеет в своем составе две функции для переключения страниц (причем, каждая из предлагаемых функции работает только в том случае, если видеоадаптер имеет в текущем режиме не менее двух страниц):

setactivpage (номер страницы) – устанавливает активную страницу экранной памяти, в которую будет осуществляться запись графического образа экрана. если номер страницы указан неверно, то весь графический вывод на эту страницу просто игнорируется.

setvisualpage (номер страницы) – устанавливает страницу, содержимое которой отображается на экране дисплея. Если был передан чрезмерно большой номер страницы, то в целом функция работает так, как если бы ее аргумент был взят по модулю "максимальное количество страниц для данного режима", однако не следует полагаться на это свойство функции, так как в этом случае возможны неприятные побочные эффекты.

При установке графического режима сразу устанавливается и графическое окно, совпадающее со всей страницей. Однако у программиста есть возможность управлять размером и расположением графического окна динамически. Делается это при помощи функции

setviewport (x левой границы, y левой границы, x правой границы, y правой страницы, p) – устанавливает размер окна экрана для вывода изображения (поле вывода), причем ни одна из этих границ не может лежать за пределами страницы. Последний аргумент устанавливает режим отсечения:

если p!=0, то с отсечением изображения за пределами графического окна;

если p=0, то сохранением изображения за пределами окна.

При успешном выполнении этой функции текущая графическая позиция перемещается в начало координат окна.

ЗАМЕЧАНИЕ. Текущая графическая позиция – графический эквивалент понятия курсора в текстовом режиме. Так графическая позиция идентифицирует выбранный пиксел графического окна, к которому привязывается действие некоторых функций, таких как вычерчивание прямолинейных отрезков или вывод графических текстов. Текущая графическая позиция характеризуется своими координатами, задаваемыми в системе графического окна. В отличие от текстового курсора текущая графическая позиция сама собой никак не отображается на экране.

При установке графического экрана функцией setviewport содержимое страницы видеопамяти не изменяется. Для очистки графического окна используется функция

clearviewport (void) – очищает ("заливает" его цветом фона) ранее установленное окно графического экрана и устанавливает текущую позицию в левый верхний угол текущего графического окна. Содержимое экрана вне текущего окна не изменяется.

В графической библиотеке имеется еще одна похожая функция

cleardevice (void) – очищает всю активную страницу. Установка графического окна при этом также не изменяется, а текущая графическая позиция перемещается в его начало координат.

Иногда бывает необходимо временно сохранить какой-либо фрагмент страницы видеопамяти с тем, чтобы в дальнейшем восстановить его на прежнем или на новом месте. Обмен производится прямоугольными массивами, задаваемыми координатами своих углов в системе координат текущего графического окна.

Прежде чем сохранять фрагмент страницы, необходимо определить требуемый для этого объем оперативной памяти. Он зависит от количества битов, приходящихся на один пиксел в текущем графическом режиме. Кроме того, вместе с содержимым массива пикселов, в память автоматически записывается и его размер. Все необходимые вычисления производит функция

imagesize (x левой границы, y верхней границы, x правой границы, y нижней границы) – возвращает число байтов памяти, необходимое для сохранения прямоугольной области экрана, заданный координатами пикселов левого верхнего и правого нижнего углов. Если для сохранения области экрана требуется более 64 Кбайт, функция возвращает 0xFFFF.

После того, как мы получили требуемый объем оперативной памяти, можно сохранить там массив пикселов при помощи функции

getimage (x левой границы, y верхней границы, x правой границы, y нижней границы, void far *bitmap) – позволяет сохранить в области памяти, на которую указывает bitmap, окно экрана с заданным размером. Кординаты сохраняемой области задаются относительно левого верхнего угла текущего графического окна. Функция помещает в буфер не только коды цветов всех пикселов, но и информацию о ширине и высоте сохраняемой области.

Для того, чтобы массив пикселов, сохраненный при помощи функции getimage снова записать на активную страницу видеопамяти, используют функцию

putimage (x левой границы, y верхней границы, void far *bitmap, операция) – выводит сохраненное изображение в окне экрана. Левый верхний угол выводимой прямоугольной области помещается в точке с координатами (x, y). Эти координаты задаются относительно левого верхнего угла текущего окна. Выводимая область экрана записана в буфере, на начало которого указывает bitmap. Операция – эта константа, определяющая способ наложения выводимого окна на другое изображение экрана. Значение этого аргумента может быть задано либо целым числом, либо символической константой.

Символическая константа	Значение	Описание стиля заполнения
COPY_PUT	0	Сохраненная ранее область экрана полностью переопределяет предыдущее его содержимое
XOR_PUT	1	Код цвета пиксела в выводимой области образуется операцией Исключающего Или кода цвета из буфера и кода цвета пиксела на экране
OR_PUT	2	Код цвета пиксела в выводимой области образуется операцией Логического Или кода цвета из буфера и кода цвета пиксела на экране
AND_PUT	3	Код цвета пиксела в выводимой области образуется операцией Логического И кода цвета из буфера и кода цвета пиксела на экране
NOT_PUT	4	Каждый бит кода цвета каждого пиксела в буфере логически инвертируется и

		после этого выводится на экран, полностью переопределяя его предыдущее содержимое

Следует обратить внимание, что операция "Исключающее Или" обладает свойством обратимости: выполненный в этом режиме дважды на одно место вывод одного и того же набора пикселов не изменяет содержимого видеобуфера, что весьма полезно при программировании движущихся по экрану графических объектов.

Сохранение изображения в окне размером 100x100 точек от начала экрана и его дальнейший вывод (в данном случае простое копирование) осуществляется следующим образом:

```
int s; void *bitmap;
s = imagesize (0, 0, 100, 100);
bitmap = malloc (s);
getimage (0, 0, 100, 100, bitmap);
putimage (300, 100, bitmap, 0);
```

ЗАМЕЧАНИЕ. Следует иметь в виду, что функция putimage выводит сохраненное ранее изображение на страницу так, как если бы никакого графического окна не существовало вовсе, то есть игнорирует режим отсечения. Если же границы массивов пикселов задавались таким образом, что весь массив или его часть выходили за пределы страницы, то результат работы функций getimage и putimage непредсказуем.

После использования вышеприведенных функций необходимо освободить память. Сделать это можно при помощи функции

graphfreemem (void far *p, размер памяти) – освобождает блок памяти заданного размера (в байтах) из области памяти, на которую указывает p; является аналогом функций free и realloc для графической библиотеки.

Приведем несколько примеров использования функций, приведенных выше.

Результатом работы первой программы является летающая тарелка, передвигающаяся по диагонали экрана, из которой появляется инопланетянин и спрашивает: " Это Томск? ".

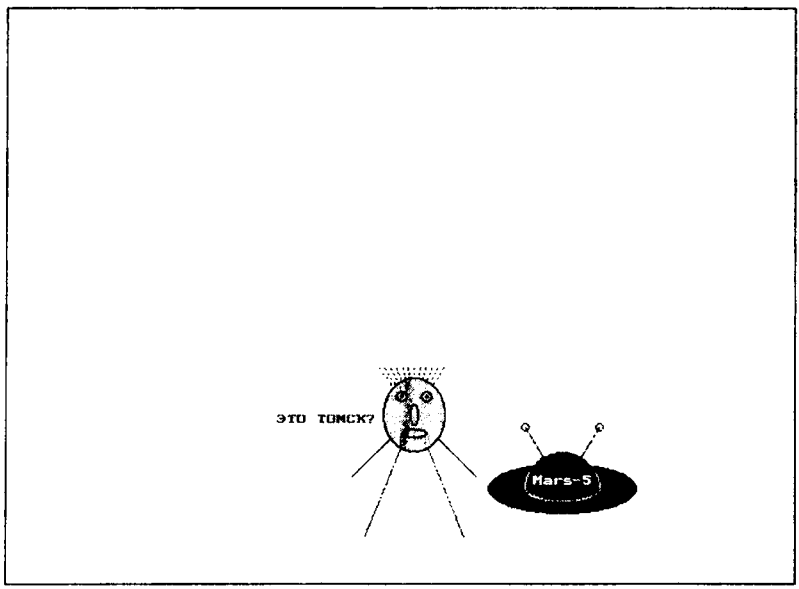
```
#include <graphics.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
```

```
main ( )
{int x1, y1, x0, y0, k, t;
 char *p;
//////////////////// инициализация графического режима ////////////////////////
 int gd=DETECT,gm;
 initgraph (&gd, &gm,"");
 cleardevice ( );                    // очистка экрана
 setbkcolor (15);                    // установка цвета фона (белый)
//////////////////////////////////// рисование "тарелки" ////////////////////////////////
 setcolor (4);                       // установка цвета линии (красный)
// установка стиля (сплошной) и цвета (красный) заливки
 setfillstyle (1, 4);
// вывели залитый установленным стилем и цветом
// эллипс – борт "тарелки"
 fillellipse (150, 150, 60, 20);
 setcolor (7);                       // сменили цвет линии на светло-серый
// нарисовали купол "тарелки"
 ellipse (150, 150, 0, 180, 30, 30);
 ellipse (150, 150, 180, 360, 30, 10);
 floodfill (150, 123, 7);            // залили купол
 setcolor (4);                       // установили красный цвет линии
// нарисовали антенны
 line (150, 150, 180, 100);
 line (150, 150, 120, 100);
 circle (180, 100, 3);
 circle (120, 100, 3);
 setcolor (15);                      // установили белый цвет линии
 outtextxy (127, 140, "Mars-5");     // на куполе вывели надпись
////////////////////////// подготовка к движению "тарелки" ////////////////////////
// определили размер "тарелки"
 k=imagesize (90, 95, 210, 180);
 p=(char *) malloc (k);              // взяли память под этот размер
 getimage (90, 95, 210, 180, p);     // запомнили рисунок в памяти
 x0=90, y0=95;
//////////////////////////////////// движение "тарелки" ////////////////////////////////////
 for (y1=x1=170; (y1<getmaxy( )-110) &&
     (x1<getmaxx( )-120); x1=x1+20, y1=y1+15)
  {
 // вывод "тарелки" в том же месте операцией "Исключающее Или",
 // то есть стерли изображение
```

```
  putimage (x0, y0, p, 1);
 // копирование изображения в другом месте
  putimage (x1, y1, p, 0); x0=x1, y0=y1;
  delay (200);                    // задержка экрана
  }
////////////////////////////////////// рисуем марсианина ////////////////////////////////
 x1-=80;
 setcolor (8);                    // установка цвета линии (темно-серый)
 setlinestyle (1, 0, 1);          // установка стиля линии (тонкая из точек)
// "прическа"
 for (x0=x1-30, y0=y1-40; x0<x1+30; x0=x0+5)     line (x1, y1, x0, y0);
 setlinestyle (0, 0, 1);          // вернули тонкую сплошную линию
// нарисовали "руки"
 line (x1, y1, x1-50, y1+50);
 line (x1, y1, x1+50, y1+50);
// нарисовали "ноги"
 line (x1, y1, x1-40, y1+100);
 line (x1, y1, x1+40, y1+100);
 setcolor (1);                    // сменили цвет линии на синий
 ellipse (x1, y1, 0, 360, 25, 30); // нарисовали "голову"
// сменили цвет заливки на светло-голубой
 setfillstyle (1, 11);
 floodfill (x1, y1, 1);           // залили "голову"
// нарисовали один "глаз"
 circle (x1-10, y1-15, 4);
 circle (x1-10, y1-15, 1);
// нарисовали другой "глаз"
 circle (x1+10, y1-15, 4);
 circle (x1+10, y1-15, 1);
 ellipse (x1, y1, 0, 360, 3, 8);        // нарисовали "нос"
 ellipse (x1, y1+15, 0, 360, 10, 3);  // нарисовали "рот"
////////////////////// подготовка к движению "марсианина" //////////////////////
// взяли память под изображение "марсианина"
 p=new char [imagesize (x1-50, y1-40, x1+50, y1+100)];
// запомнили рисунок в памяти
 getimage (x0=x1-50, y0=y1-40, x1+50, y1+100, p);
////////////////////////////////// движение "марсианина" ////////////////////////////////
 for (k=0, t=-20; k<7; k++)          // прыгает 7 раз
  {// стерли изображение операцией "Исключающее Или"
   putimage (x0, y0, p, 1);
```

```
    // вывели изображение в другом месте
    putimage (x0, y0=y0+t, p, 0); t=-t;
   delay (500);                    // задержка экрана
  }
 setcolor (5);                  // сменили цвет линии на фиолетовый
// успокаивается и спрашивает: "ЭТО ТОМСК?"
 outtextxy (x1-110, y1-20, "ЭТО ТОМСК?");
}
```

Заканчивается работа этой программы следующей картинкой:



Результатом работы второй программы является "звездное небо". Вначале появляются разноцветные точки ("мелкие звезды") и одна более крупная звезда. Затем эта звезда случайным образом "размножается" на экране до нажатия клавиши Enter.

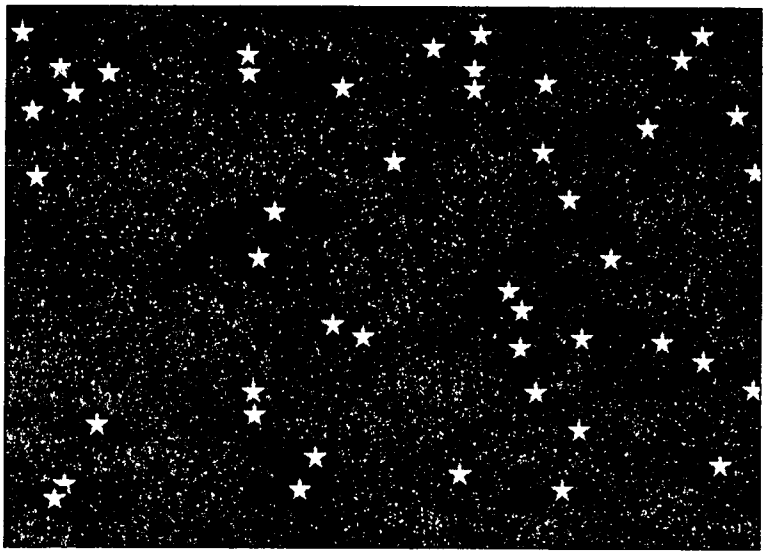
```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
```

```
#include <stdlib.h>
#include <alloc.h>
main ( )
{int *m,k,r,i,*p;
 float a,da;
  m=(int *) malloc (12*sizeof (int));
  da = M_PI * 4 / 5; a=1.57; r=10;
////////////////////// формирование 5-тиконечной звезды //////////////////////////
  for (k=0; k<12;)
    {m [k++]=(int) (r*cos (a))+50;
     m [k++]=-(int) (r*sin (a))+50;
     a=a+da;
    }
////////////////////// инициализация графической системы //////////////////////////
 int gd=DETECT, gm=3;
 initgraph (&gd, &gm, "");
// установка цвета фона и цвета линии (желтый)
 setcolor (14); setbkcolor (14);
////////////////////////////////////// вывод мелких звезд //////////////////////////////////
 randomize ( );k=0;
 while (!kbhit ( ))                    // пока не нажата любая клавиша
  {k++;
  // выводим точку, координаты которой случайные числа,
  // всевозможными цветами
   putpixel (random (getmaxx ( )), random (getmaxy ( )), k%16);
  }
 getchar();                            // задержка экрана
////////////////////////////////////// вывод крупных звезд //////////////////////////////////
 drawpoly (6, m);                      // вывели звезду
// залили ее сплошным желтым цветом
 setfillstyle (1, 14); floodfill (1, 1, 14);
// взяли память под звезду
 p=(int *) malloc (imagesize (40, 40, 60, 60));
 getimage (40, 40, 60, 60, p);         // запомнили звезду
 while (!kbhit ( ))                    // пока не нажата любая клавиша
  {// вывели сохраненное изображение в режиме копирования
   // в точку со случайными координатами
   putimage (random (getmaxx ( )-20), random (getmaxy ( )-20), p, 0);
   delay(200);                         // задержка экрана
  }
```

```
getchar();getchar();                    // задержка экрана

}
```

Результат работы этой программы выглядит следующим образом:



Рассматриваемые функции графической библиотеки Turbo C

В приложении приводится полный синтаксис описания функций графической библиотеки в алфавитном порядке с указанием страницы, где содержится полное описание каждой функции.

Тип функции	Имя функции	Номер страницы
int	getx (void)	7
int	gety (void)	7
void	graphfreemem (void far *p, unsigned размер памяти)	34
int	graphresult (void)	4
unsigned	imagesize (int x левой границы, int y верхней границы., int x правой границы, int y нижней границы)	32
void	initgraph (int &gd, int &gm, char *pathtodriver)	4
void	line (int x начала, int y начала, int x конца, int y конца)	6
void	linerel (int x приращения, int y приращения)	6
void	lineto (int x, int y)	6
void	moverel (int x приращения, int y приращения)	6
void	moveto (int x, int y)	6
void	outtext (char *textstring)	14
void	outtextxy (int x, int y, char *textstring)	14
void	pieslice (int x центра, int y центра, int угол начала, int угол конца, int радиус)	26
void	putimage (void)	33
void	putpixel (int x, int y, int цвет)	6
void	rectangle (int x левой стороны, int y верхней стороны, int x правой стороны, int y нижней стороны)	6
void	sector (int x центра, int y центра, int угол начала, int угол конца, int горизонтальный радиус, int вертикальный радиус)	26
void	setactivpage (int номер страницы)	31
void	setbkcolor (int номер цвета)	20
void	setcolor (int номер цвета)	20
void	setfillpattern (char *up, int номер цвета)	25
void	setfillstyle (int номер наполнителя, int номер цвета)	21
void	setlinestyle (int стиль линии, unsigned образец битового базового отрезка, int толщина)	26
void	settextjustify (int по горизонтали, int по вертикали)	13
void	settextstyle (int шрифт, int направление, int размер символа)	11
void	setusercharsize (int multx, int divx, int multy, int divy)	12
void	setviewport (int x левой стороны, int y верхней стороны, int x правой стороны, int y нижней стороны, int p)	31

Литература

1. Касаткин А. И. Управление ресурсами. – Минск: "Вышейшая школа", 1992. – 431 с.

2. Скляров В. А. Программное и лингвистическое обеспечение персональных ЭВМ. Системы общего назначения. – Минск: "Вышейшая школа", 1992. – 462 с.

3. Прокофьев Б. П., Сухарев Н. Н., Храмов Ю.Е. Графические средства Turbo C и Turbo C++. – Москва: "Финансы и статистика", 1992. – 159 с.

СОДЕРЖАНИЕ

ОСНОВЫ ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКОЙ БИБЛИОТЕКИ ЯЗЫКА СИ И СИ++. Учебное пособие. /Томский государственный университет. – Томск, 2000. – 45с.

Подп. в печать Тираж 150 экз.

Заказ № УОП ТГУ, Томск ул.