CHƯƠNG 8

ĐỒ HỌA

Trong chương này sẽ giới thiệu các hàm để vẽ các đường và hình cơ bản như đường tròn, cung elip, hình quạt, đường gẫy khúc, hình đa giác, đường thẳng, đường chữ nhật, hình chữ nhật, hình hộp chữ nhật, ... Ngoài ra còn đề cập tới các vấn đề rất lý thú khác như: xử lý văn bản trên màn hình đồ họa, cửa sổ và kỹ thuật tạo ảnh di động. Các hàm đồ họa được khai báo trong tệp graphics.h.

§ 1. KHÁI NIỆM ĐỒ HỌA

Để hiểu kỹ thuật lập trình đồ họa, đầu tiên phải hiểu các yếu tố cơ bản của đồ họa. Từ trước đến nay chúng ta chủ yếu làm việc với kiểu văn bản. Nghĩa là màn hình được thiết lập để hiển thị 25 dòng, mỗi dòng có thể chứa 80 ký tự. Trong kiểu văn bản, các ký tự hiển thị trên màn hình đã được phần cứng của máy PC ấn định trước và ta không thể nào thay đổi được kích thước, kiểu chữ.

Ở màn hình đồ họa, ta có thể xử lý đến từng chấm điểm (pixel) trên màn hình và do vậy muốn vẽ bất kỳ thứ gì cũng được. Sự bài trí và số pixel trên màn hình được gọi là độ phân giải (resolution). Do mỗi kiểu màn hình đồ họa có một cách xử lý đồ họa riêng nên TURBO C cung cấp một tệp tin điều khiển riêng cho từng kiểu đồ họa. Bảng 8-1 cho thấy các kiểu đồ họa và các tệp tin điều khiển chúng.

Ngoài các tệp có đuôi BGI chứa chương trình điều khiển đồ họa, TURBO C còn cung cấp các tệp tin đuôi CHR chứa các Font chữ để vẽ các kiểu chữ khác nhau trên màn hình đồ họa. Đó là các tệp:

GOTH.CHR

LITT.CHR

SANS.CHR

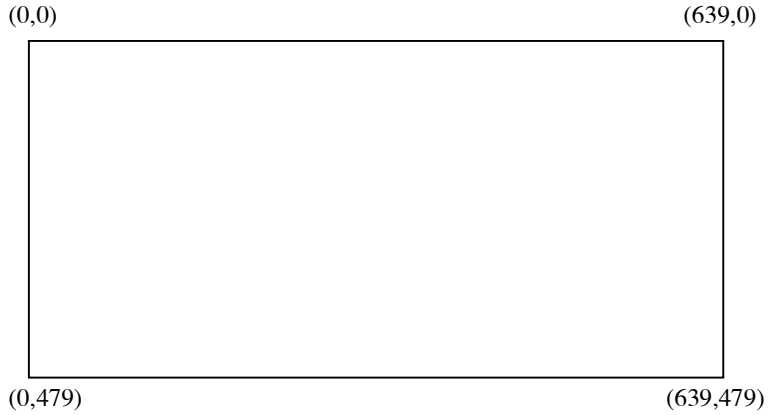
TRIP.CHR

Bảng 8-1. Các tệp tin điều khiển đồ họa của TURBO C++

Tên tệp tin	Kiểu màn hình đồ họa
ATT.BGI	ATT & T6300 (400 dòng)
CGA.BGI	IBMCGA, MCGA và các máy tương thích
EGAVGA.BGI	IBM EGA, VGA và các máy tương thích
HERC.BGI	Hercules monochrome và các máy tương thích
IBM8514.BGI	IBM 8514 và các máy tương thích
PC3270.BGI	IBM 3270 PC

Màn hình đồ họa gồm nhiều điểm ảnh được sắp xếp trên các đường thẳng ngang và dọc. Điều này đúng cho tất cả các kiểu màn hình đồ họa của máy tính. Khác biệt chủ yếu giữa chúng là kích thước và số các điểm ảnh. Trong kiểu CGA (độ phân giải thấp), điểm ảnh có kích thước lớn, chiều ngang có 320 điểm ảnh, còn theo chiều dọc có 200 điểm ảnh. Màn hình VGA có độ phân giải cao hơn: điểm ảnh nhỏ hơn, trên mỗi hàng có 640 điểm ảnh và trên mỗi cột có 480 điểm ảnh. Điểm ảnh càng nhỏ thì số điểm ảnh trên màn hình càng nhiều và chất lượng đồ họa càng cao.

Mỗi kiểu đồ họa dùng một hệ tọa độ riêng. Hệ tọa độ cho màn hình VGA là 640 x 480 như sau :



Hình 8.1. Hệ tọa độ VGA

Nhờ hệ tọa độ này, ta có thể tác động hay tham chiếu đến bất kỳ điểm ảnh nào trên màn hình đồ họa.

Nếu dùng màn hình CGA thì góc dưới phải có tọa độ (319, 199). Độc lập với kiểu đồ họa đang sử dụng, các hàm getmaxx và getmaxy bao giờ cũng cho tọa độ x và y lớn nhất trong kiểu đồ họa đang dùng.

Một chương trình đồ họa thường gồm các phần sau:

- Khởi động hệ thống đồ họa.

- Xác định mầu nền (mầu màn hình), mầu đường vẽ, mầu tô và kiểu (mẫu) tô.

- Vẽ, tô mầu các hình mà ta mong muốn.

- Các thao tác đồ họa khác như cho hiện các dòng chữ...

- Đóng hệ thống đồ họa để trở về mode văn bản.

§ 2. KHỞI ĐỘNG HỆ ĐỒ HỌA

Mục đích của việc khởi động hệ thống đồ họa là xác định thiết bị đồ họa (màn hình) và mốt đồ họa sẽ sử dụng trong chương trình. Để làm điều này ta dùng hàm:

```
void initgraph(int *graphdriver, int *graphmode,char *driverpath);
```

trong đó: driverpath là đường dẫn của thư mục chứa các tệp tin điều khiển đồ họa, graphdriver, graphmode cho biết màn hình và mốt đồ họa sẽ sử dụng trong chương trình. Bảng 8-2 cho thấy các giá trị khả dĩ của graphdriver và graphmode.

Ví dụ 1. Giả sử máy tính của ta có màn hình EGA, các tệp tin đồ họa chứa trong thư mục C: \TC, khi đó ta có thể khởi động hệ thống đồ họa như sau:

```
#include "graphics.h"
main()
  {
    int mh=EGA, mode= EGALO;
    initgraph(&mh, &mode, "C:\TC");
    . . .
  }
```


Bảng 8-2. Các giá trị khả dĩ của graphdriver, graphmode

graphdriver	graphmode	Độ phân giải
Detect (0)		
CGA (1)	CGAC0 (0)	320 x 200
	CGAC1 (1)	320 x 200
	CGAC2 (2)	320 x 200
	CGAC3 (3)	320 x 200
	CGAHi (4)	640 x 200
MCGA (2)	MCGA0 (0)	320 x 200
	MCGA1 (1)	320 x 200
	MCGA2 (2)	320 x 200
	MCGA3 (3)	320 x 200
	MCGAMed (4)	640 x 200
	MCGAHi (5)	640 x 480
EGA (3)	EGALO (0)	640 x 200
	EGAHi (1)	640 x 350
EGA64 (4)	EGA64LO (0)	640 x 200
	EGA64Hi (1)	640 x 350
EGAMONO (5)	EGAMONOHi (0)	640 x 350
VGA (9)	VGALO (0)	640 x 200
	VGAMED (1)	640 x 350
	VGAHI (2)	640 x 480
HERCMONO (7)	HERCMONOHI	720 x 348
ATT400 (8)	ATT400C0 (0)	320 x 200
	ATT400C1 (1)	320 x 200
	ATT400C2 (2)	320 x 200
	ATT400C3 (3)	320 x 200
	ATT400MED (4)	640 x 400
	ATT400HI (5)	640 x 400
PC3270 (10)	PC3270HI (0)	720 x 350
IBM8514 (6)	IBM8514LO (0)	640 x 480, 256 mầu
	IBM8514HI (1)	1024 x 768, 256 mầu

Chú ý 1. Bảng 8-2 cho các tên hằng và giá trị của chúng mà các biến graphdriver, graphmode có thể nhận. Chẳng hạn hằng DETECT có giá trị 0, hằng VGA có giá trị 9, hằng VGALO có giá trị 0... Khi lập trình ta có thể dùng tên hằng hoặc giá trị tương ứng của chúng. Chẳng hạn các phép gán trong ví dụ 1 có thể viết theo một cách khác tương đương như sau:

```
mh=3;
mode=0;
```

Chú ý 2. Bảng 8.2 cho thấy độ phân giải phụ thuộc cả vào màn hình và mode. Ví dụ trong màn hình EGA nếu dùng mode EGALO thì độ phân giải là 640 x 200, hàm getmaxx cho giá trị 639, hàm getmaxy cho giá trị 199 . Nếu cũng màn hình EGA mà dùng mode EGAHI thì độ phân giải là 640x 350, hàm getmaxx cho giá trị 639, hàm getmaxy cho giá trị 349.

Chú ý 3. Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver hằng DETECT hay giá trị 0. Khi đó kết quả của hàm initgraph sẽ là:

- Kiểu của màn hình đang sử dụng được phát hiện, giá trị số của nó được gán cho biến graphdriver.

- Mode đồ họa ở độ phân giải cao nhất ứng với màn hình đang sử dụng cũng được phát hiện và giá trị số của nó được gán cho biến graphmode.

Như vậy việc dùng hằng số DETECT chẳng những có thể khởi động được hệ thống đồ họa của màn hình hiện có theo mode có độ phân giải cao nhất, mà còn giúp ta xác định chính xác kiểu màn hình đang sử dụng.

Ví dụ 2. Chương trình dưới đây xác định kiểu màn hình đang sử dụng:

```
#include "graphics.h"
#include "stdio.h"
main()
  {
    int mh=0, mode= 0;
    initgraph(&mh, &mode, "");
    printf("\n Giá trị số của màn hình là: %d", mh);
    closegraph();
  }
```

Nếu chương trình cho kết quả:

Giá trị số của màn hình là: 3

thì ta có thể khẳng định loại màn hình đang dùng là EGA.

Chú ý 4. Nếu chuỗi dùng để xác định driverpath là một chuỗi rỗng (như trong ví dụ 2) thì chương trình dịch sẽ tìm các tệp điều khiển đồ họa trên thư mục chủ.

§ 3. LỖI ĐỒ HỌA

Khi khởi động hệ thống đồ họa nếu máy không tìm thấy các chương trình điều khiển đồ họa thì sẽ phát sinh lỗi đồ họa và việc khởi động coi như không thành. Lỗi đồ họa còn phát sinh khi dùng các hàm đồ hoạ. Trong mọi trường hợp, hàm graphresult cho biết có lỗi hay không lỗi và đó là lỗi gì. Bảng 8-3 cho các mã lỗi mà hàm này phát hiện được. Ta có thể dùng hàm grapherrormsg với mã lỗi do hàm graphresult trả về để biết được đó là lỗi gì, ví dụ:

```
int maloi;
maloi = graphresult();
printf("\nLỗi đồ họa là: %d", grapherrormsg(maloi));
```

Bảng 8-3. Các mã lỗi của Graphresult

Hằng	Trị	Lỗi phát hiện
grOk	0	Không có lỗi
grNoInitGraph	-1	Chưa khởi động hệ đồ họa
grNotDetected	-2	Không có phần cứng đồ họa
grFileNotFound	-3	Không tìm thấy trình điều khiển đồ họa
grInvalidDriver	-4	Trình điều khiển không hợp lệ
grNoLoadMem	-5	Không đủ RAM cho đồ họa

grNoScanMem	-6	Vượt vùng RAM trong Scan fill
grNoFloodMem	-7	Vượt vùng RAM trong flood fill
grFontNoFound	-8	Không tìm thấy tập tin Font
grNoFontMem	-9	Không đủ RAM để nạp Font
grInvalidMode	-10	Kiểu đồ họa không hợp lệ cho trình điều khiển
grError	-11	Lỗi đồ họa tổng quát
grIOerror	-12	Lỗi đồ họa vào ra
grInvalidFont	-13	Tập tin Font không hợp lệ
grInvalidFontNum	-14	Số hiệu Font không hợp lệ

§ 4. MẦU VÀ MẪU

1. Để chọn mầu nền ta sử dụng hàm

void setbkcolor(int color);

2. Để chọn mầu đường vẽ ta dùng hàm

void setcolor(int color);

3. Để chọn mẫu (kiểu) tô và mầu tô ta dùng hàm

void setfillstyle(int pattern, int color);

Trong cả 3 trường hợp color xác định mã của mầu. Các giá trị khả dĩ của color cho trong bảng 8-4, pattern xác định mã của mẫu tô (xem bảng 8-5).

Mẫu tô và mầu tô sẽ được sử dụng trong các hàm pieslice, fillpoly, bar, bar3d và floodfill (xem §5 dưới đây).

4. Chọn giải mầu

Để thay đổi giải mầu đã được định nghĩa trong bảng 8.4 ta dùng hàm

void setpalette(int colornum, int color);

Ví dụ câu lệnh

setpalette(0, LIGHTCYAN);

biến mầu đầu tiên trong bảng mầu thành xanh lơ nhạt. Các mầu khác không bị ảnh hưởng.

Bảng 8-4. Các giá trị khả dĩ của color

Tên hằng	Giá trị số	Mầu hiển thị
BLACK	0	Đen
BLUE	1	Xanh da trời
GREEN	2	Xanh lá cây
CYAN	3	Xanh lơ
RED	4	Đỏ
MAGENTA	5	Tím
BROWN	6	Nâu
LIHGTGRAY	7	Xám nhạt
DARKGRAY	8	Xám sẫm
LIGHTBLUE	9	Xanh da trời nhạt
LIGHTGREEN	10	Xanh lá cây nhạt
LIGHTCYAN	11	Xanh lơ nhạt
LIGHTRED	12	Đỏ nhạt
LIGHTMAGENTA	13	Tím nhạt
YELLOW	14	Vàng
WHITE	15	Trắng

5. Để nhận giải mầu hiện hành ta dùng hàm

void getpalette (struct palettetype *palette);

ở đây palettetype là kiểu đã định nghĩa trước như sau:

```
#define MAXCOLORS 15
struct palettetype
  {
    unsigned char size;
    unsigned char colors[MAXCOLORS+1];
  };
```

ở đây: size là số lượng mầu trong palette, colors là mảng chứa mầu với chỉ số mảng chạy từ 0 đến size - 1

Bảng 8-5. Các giá trị khả dĩ của pattern

Tên hằng	Giá trị số	Mô tả kiểu tô
EMPTY_FILL	0	Tô bằng mầu nền
SOLID_FILL	1	Tô bằng đường nét liền
LINE_FILL	2	Tô bằng - - -
LTSLASH_FILL	3	Tô bằng ///
SLASH_FILL	4	Tô bằng /// in đậm
BKSLASH_FILL	5	Tô bằng \\\ in đậm
LTBKSLASH_FILL	6	Tô bằng \\\
HATCH_FILL	7	Tô bằng đường gạch bóng nhạt
XHATCH_FILL	8	Tô bằng đường gạch bóng chữ thập
INTERLEAVE_FILL	9	Tô bằng đường đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm thưa
CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

6. Hàm getcolor trả về mầu đã xác định trước đó bằng hàm setcolor.

7. Hàm getbkcolor trả về mầu đã xác định trước đó bằng hàm setbkcolor.

8. Hàm getmaxcolor trả về mã mầu cực đại thuộc giải mầu hiện đang có hiệu lực. Trên 256 K EGA, hàm getmaxcolor luôn cho giá trị 15.

§ 5. VẼ VÀ TÔ MẦU

Có thể chia các đường và hình thành bốn nhóm chính:

- Đường tròn và ellipse
- Đường gấp khúc và hình đa giác
- Đường thẳng
- Hình chữ nhật

A. Đường tròn và hình tròn

Nhóm này gồm cung tròn, đường tròn, cung ellipse và hình quạt.

1. Cung tròn. Để vẽ một cung tròn ta dùng hàm

```
void arc(int x, int y, int gd, int gc, int r);
```

ở đây:

(x, y) là tọa độ của tâm cung tròn,
r là bán kính
gd là góc đầu
gc là góc cuối

Chú ý: Trong tất cả các hàm dưới đây, góc tính theo độ và có giá trị từ 0 đến 360.

2. Đường tròn. Để vẽ một đường tròn ta dùng hàm

```
void circle(int x, int y, int r);
```

ở đây:

(x, y) là tọa độ của tâm;
r là bán kính đường tròn.

3. Cung ellipse. Để vẽ một cung Ellipse ta dùng hàm

```
void ellipse(int x,int y,int gd,int gc,int xr,int yr);
```

ở đây:

(x, y) là tọa độ của tâm cung Ellipse
gd là góc đầu
gc là góc cuối
xr là bán trục ngang
yr là bán trục đứng.

4. Hình quạt. Để vẽ và tô màu một hình quạt ta dùng hàm

```
void pieslice(int x,int y,int gd,int gc,int r);
```

ở đây:

(x,y) là tọa độ tâm hình quạt
gd là góc đầu

gc là góc cuối

r là bán kính

Ví dụ 1. Chương trình dưới đây sẽ vẽ: một cung tròn ở góc phần tư thứ nhất, một cung ellipse ở góc phần tư thứ ba, một đường tròn và một hình quạt quét từ 90 đến 360 độ.

```
#include <graphics.h>
main()
  {
    int mh, mode;
    // Khởi động đồ họa, màn hình EGA, mode EGALO
    mh=EGA;
    mode=EGALO;
    initgraph(&mh, &mode,"");
    // Mầu nền Green, mầu đường vẽ
    //White, mầu tô Red, kiểu tô SlashFill
    setbkcolor (GREEN);
    setcolor (WHITE);
    setfillstyle (SLASH_FILL, RED);
    // Vẽ: một cung tròn ở góc phần tư thứ nhất,
    // một cung Ellipse ở góc phần tư thứ ba,
    // một đường tròn, một quạt tròn
    arc(160, 50, 0, 90, 45);
    ellipse(480, 50, 180, 270, 150, 45);
    circle(160, 150, 45);
    pieslice(480, 150, 90, 360, 45);
    // Kết thúc chế độ đồ họa
    closegraph();
  }
```


B. Đường gấp khúc và đa giác

5. Muốn vẽ một đường gấp khúc đi qua n điểm: (x1,y1), ... , (xn,yn) thì trước hết ta phải đưa các tọa độ vào một mảng a nào đó kiểu int. Nói một cách chính xác hơn, cần gán x1 cho a[0], y1 cho a[1], x2 cho a[2], y2 cho a[3],... Sau đó ta viết lời gọi hàm:

```
drawpoly(n, a);
```

Khi điểm cuối (xn, yn) trùng với điểm đầu (x1, y1) ta nhận được một đường gấp khúc khép kín.

6. Giả sử a là mảng đã nói trong điểm 5, khi đó lời gọi hàm

```
fillpoly(n, a);
```

sẽ vẽ và tô mầu một đa giác có đỉnh là các điểm

(x1, y1), ... ,(xn, yn).

Ví dụ 2. Chương trình dưới đây sẽ vẽ một đường gấp khúc và hai hình tam giác.

```
#include <graphics.h>
// Xây dựng các mảng chứa tọa độ các đỉnh
int poly1[]={5,200,190,5,100,300};
int poly2[]={205,200,390,5,300,300};
int poly3[]={405,200,590,5,500,300,405,200};
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    // Mầu nền CYAN, mầu đường vẽ
    // YELLOW, mầu tô MAGENTA, mẫu tô SolidFill
    setbkcolor (CYAN); Setcolor (YELLOW);
    setfillstyle (SOLID_FILL, MAGENTA);
    drawpoly (3, poly1); // Đường gấp khúc
    fillpoly (3, poly2); // Hình đa giác
```

```
    fillpoly(4, poly3); // Hình đa giác
    closegraph();
  }
```

C. Đường thẳng

7. Hàm

```
void line(int x1,int y1,int x2,int y2);
```

vẽ đường thẳng nối hai điểm (x1, y1) và (x2, y2) nhưng không làm thay đổi vị trí con chạy.

8. Hàm

```
void lineto(int x,int y);
```

vẽ đường thẳng từ điểm hiện tại tới điểm (x, y) và chuyển con chạy đến điểm (x, y).

9. Hàm

```
void linerel(int dx,int dy);
```

vẽ một đường thẳng từ vị trí hiện tại (x, y) của con chạy đến điểm (x + dx,y + dy). Con chạy được di chuyển đến vị trí mới.

10. Hàm

```
void moveto(int x,int y);
```

sẽ di chuyển con chạy tới vị trí (x, y).

Ví dụ 3. Chương trình dưới đây tạo lên một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh: (20, 20), (620, 20), (620, 180), (20, 180) và (320, 100).

```
#include <graphics.h>
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    setbkcolor(GREEN);
```



```
    setcolor(YELLOW);
    moveto(320,100);
    line(20,20,620,20);
    linerel(-300,80);
    lineto(620,180);
    lineto(620,20);
    closegraph();
  }
```

D. Hình chữ nhật

11. Hàm

```
void rectangle(int x1,int y1,int x2,int y2);
```

sẽ vẽ một đường chữ nhật có các cạnh song song với các cạnh của màn hình. Tọa độ đỉnh trên bên trái của hình chữ nhật là (x1,y1) và điểm dưới bên phải là (x2,y2).

12. Hàm

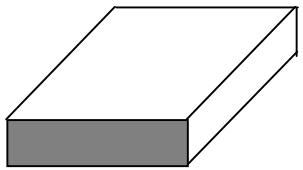
```
void bar(int x1,int y1,int x2,int y2);
```

sẽ vẽ và tô mầu một hình chữ nhật. Các giá trị x1,y1,x2 và y2 có ý nghĩa như đã nói trong điểm 11.

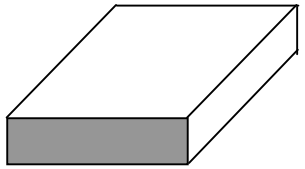
13. Hàm

```
void bar3d(int x1,int y1,int x2,int y2,int depth,int top);
```

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các tọa độ x1,y1,x2,y2 (như đã nói trong điểm 12). Hình chữ nhật này được tô mầu. Tham số depth ấn định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số top có thể nhận trị 1 (TOPON) hay 0 (TOPOFF) và khối 3 chiều sẽ có nắp hay không nắp (xem hình vẽ).



TOPON



TOPOFF

Ví dụ 4. Chương trình dưới đây sẽ vẽ một đường chữ nhật, một hình chữ nhật và một khối hộp chữ nhật có nắp.

```
#include <graphics.h>
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(CLOSE_DOT_FILL,YELLOW);
    rectangle(5,5,300,160);
    bar(5,175,300,340);
    bar3d(320,100,500,340,100,1);
    closegraph();
  }
```

§ 6. CHỌN KIỂU ĐƯỜNG

1. Hàm

```
void setlinestyle(int linestyle,int pattern,int thickness);
```

tác động đến nét vẽ của các thủ tục line, lineto, rectange, drawpoly, circle,... Hàm này cho phép ta ấn định 3 yếu tố của đường thẳng là dạng, bề dầy và mẫu tự tạo.

+ Dạng đường do tham số linestyle khống chế. Sau đây là các giá trị khả dĩ của linestyle và dạng đường thẳng tương ứng.

SOLID_LINE = 0	Nét liền
DOTTED_LINE = 1	Nét chấm
CENTER_LINE = 2	Nét chấm gạch
DASHED_LINE = 3	Nét gạch
USERBIT_LINE = 4	Mẫu tự tạo

+ Bề dầy do tham số thickness khống chế. Giá trị này có thể là:

NORM_WIDTH = 1	Bề dầy bình thường
THICK_WIDTH = 3	Bề dầy gấp ba

+ Mẫu tự tạo: Nếu tham số thứ nhất là USERBIT_LINE thì ta có thể tạo ra mẫu đường thẳng bằng tham số pattern. Ví dụ xét đoạn chương trình:

```
int pattern= 0x1010;
setlinestyle(USERBIT_LINE, pattern, NORM_WIDTH);
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 0x1010 hay trong hệ 2 là

0001 0000 0001 0000

Chỗ nào có bit 1 điểm ảnh sẽ sáng, bit 0 làm tắt điểm ảnh.

2. Để nhận các giá trị hiện hành của 3 yếu tố trên ta dùng hàm:

```
void getlinesettings(struct linesettingstype *lineinfo);
```

với kiểu linesettingstype đã được định nghĩa trước như sau:

```
struct linesettingstype
  {
    int linestyle;
    unsigned int upattern;
    int thickness;
  };
```

Ví dụ 1. Chương trình dưới đây minh họa cách dùng các hàm setlinestyle và getlinesettings để vẽ đường thẳng.

```
// kiểu đường
#include <graphics.h>
#include <conio.h>
main()
  {
    struct linesettingstype kieu_cu;
```

```
int mh=0, mode=0;
initgraph(&mh, &mode, "");
if (graphresult!= grOk) exit(1);
setbkcolor(GREEN); setcolor(RED);
line(0,0,100,0);
// Lưu lại kiểu hiện tại
getlinesettings(kieu_cu);
// Thiết lập kiểu mới
setlinestyle(DOTTED_LINE,0,THICK_WIDTH);
line(0,0,100,10);
// Phục hồi kiểu cũ
setlinestyle(kieu_cu.linestyle,
kieu_cu.upattern, kieu_cu.thickness);
Line(0,20,100,20);
getch();
closegraph();
}
```

3. Hàm

void setwritemode(int writemode);

sẽ thiết lập kiểu thể hiện đường thẳng cho các hàm line, drawpoly, linerel, lineto, rectangle. Kiểu thể hiện do tham số writemode khống chế:

- Nếu writemode bằng COPY_PUT = 0, thì đường thẳng được viết đè lên dòng đang có trên màn hình.

- Nếu writemode bằng XOR_PUT = 1, thì mầu của đường thẳng định vẽ sẽ kết hợp với mầu của từng chấm điểm của đường hiện có trên màn hình theo phép toán XOR (chương 3, §3) để tạo lên một đường thẳng mới.

Một ứng dụng của XOR_PUT là: Khi thiết lập kiểu writemode bằng XOR_PUT rồi vẽ lại đường thẳng cùng mầu thì sẽ xóa đường thẳng cũ và khôi phục trạng thái của màn hình.

Chương trình dưới đây minh họa cách dùng hàm setwritemode. Khi thực hiện ta sẽ thấy hình chữ nhật thu nhỏ dần vào tâm màn hình.

Ví dụ 2:

```
// Thu hình;
#include <graphics.h>
#include <conio.h>
main()
 {
  struct linesettingstype kieu_cu;
  int mh=0, mode=0, x1, y1, x2, y2;
  initgraph(&mh, &mode, "");
  if (graphresult!= grOk) exit(1);
  setbkcolor(GREEN);
  setcolor(RED);
  setfillstyle(CLOSE_DOT_FILL, YELLOW);
  x1=0; y1=0;
  x2=getmaxx(); y2=getmaxy();
  setwritemode(XOR_PUT);
  tt: rectangle(x1,y1,x2,y2); // Vẽ hình chữ nhật
  if ( (x1+1)<(x2-1) && (y1+1)<(y2-1) )
   {
    rectangle(x1,y1,x2,y2); // xóa hình chữ nhật
    x1=x1+1; y1=y1+1; co hình chữ nhật
    x2=x2-1; y2=y2-1;
    goto tt;
   }
  setwritemode(COPY_PUT); // Trở về overwrite mode
  closegraph();
}
```

§ 7. CỬA SỔ (VIEWPORT)

1. Viewport là một vùng chữ nhật trên màn hình đồ họa tựa như window trong textmode. Để thiết lập viewport ta dùng hàm

```
void setviewport(int x1,int y1,int x2,int y2,int clip);
```

trong đó (x1,y1) là tọa độ góc trên bên trái và (x2,y2) là tọa độ góc dưới bên phải. Bốn giá trị này phải thỏa mãn:

```
0 <= x1 <= x2
0 <= y1 <= y2
```

Tham số clip có thể nhận một trong hai giá trị:

clip = 1 không cho phép vẽ ra ngoài viewport

clip = 0 Cho phép vẽ ra ngoài viewport.

Ví dụ câu lệnh

```
setviewport(100,50,200,150, 1);
```

sẽ thiết lập viewport. Sau khi lập viewport ta có hệ tọa độ mới mà góc trên bên trái của viewport sẽ có tọa độ (0,0).

2. Để nhận viewport hiện hành ta dùng hàm

```
void getviewsettings(struct viewporttype *vp);
```

ở đây kiểu viewporttype đã được định nghĩa như sau:

```
struct viewporttype
   {
     int left, top, right, bottom;
     int clip;
   };
```

3. Để xóa viewport ta dùng hàm

```
void clearviewport(void);
```

4. Để xóa màn hình và đưa con chạy về tọa độ (0,0) của màn hình ta dùng hàm

```
void cleardevice(void);
```

Chú ý: Câu lệnh này sẽ xóa mọi thứ trên màn hình.

5. Tọa độ âm dương

Nhờ sử dụng Viewport có thể viết các chương trình đồ họa theo tọa độ âm dương. Muốn vậy ta thiết lập viewport sao cho tâm tuyệt đối của màn hình là góc trên bên trái của viewport và cho clip = 0 để có thể vẽ ra ngoài giới hạn của viewport. Sau đây là đoạn chương trình thực hiện công việc trên

```
int xc, yc;
xc= getmaxx()/2; yc= getmaxy()/2;
setviewport(xc, yc, getmaxx(), getmaxy(), 0);
```

Như thế màn hình sẽ được chia làm 4 phần với tọa độ âm dương như sau:

Phần tư trái trên: x âm, y âm

Phần tư trái dưới: x âm, y dương

Phần tư phải trên: x dương, y âm

Phần tư phải dưới: x dương, y dương

Chương trình dưới đây vẽ đồ thị hàm sin(x) trong hệ trục tọa độ âm dương. Hoành độ x lấy các giá trị từ -4*PI đến 4*PI. Trong chương trình có dùng hai hàm mới là: outtextxy và putpixel (xem các mục sau).

Ví dụ 1:

```
// đồ thị hàm sin
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define SCALEX 20
#define SCALEY 60
main()
  {
    int mh=0, mode=0, x, y, i;
    initgraph(&mh, &mode, "");
```

```
if (graphresult!= grOk) exit(1);
setviewport(getmaxx()/,getmaxy()/2,
getmaxx(),getmaxy(), 0);
// Kẻ hệ trục tọa độ
setcolor(BLUE);
line(-(getmaxx()/2),0,getmaxx()/2,0);
line(0,-(getmaxy()/2),0,getmaxy()/2);
settextjustify(1,1); setcolor(RED);
outtextxy(0,0,"(0,0)");
for (i=-400;i<=400;++i)
  {
    x=round(2*M_PI*i*SCALEX/200);
    y=round(sin(2*M_PI*i/200)*SCALEY);
    putpixel(x,y,YELLOW);
  }
getch();
}
```

Ví dụ 1 tạo lên một đồ thị từ các chấm điểm. Bây giờ ta sửa ví dụ 1 đôi chút: giữ nguyên từ đầu đến outtextxy, thay phần cuối bởi đoạn chương trình dưới đây. Ta sẽ được đồ thị từ các đoạn thẳng rất ngắn ghép lại.

Ví dụ 2:

```
// Phần đầu giống ví dụ 1
setcolor(YELLOW);
for (i=-400;i<=400;++i)
  {
    x=round(2*M_PI*i*SCALEX/200);
    y=round(sin(2*M_PI*i/200)*SCALEY);
      if(i= -400) moveto(x,y);
      else lineto(x,y);
  }
 getch();
}
```


§ 8. TÔ ĐIỂM, TÔ MIỀN

1. Hàm

```
void putpixel(int x, int y, int color);
```

sẽ tô điểm (x,y) theo mầu xác định bởi color.

2. Hàm

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu mầu của điểm ảnh ở vị trí (x,y). Chú ý: nếu điểm này chưa được tô mầu bởi các hàm vẽ hoặc putpixel (mà chỉ mới được tạo mầu nền bởi setbkcolor) thì hàm cho giá trị bằng 0. Vì vậy có thể dùng hàm này theo mẫu dưới đây để xác định các nét vẽ trên màn hình đồ hoạ và vẽ ra giấy.

```
if (getpixel(x,y)!=0)
  {
    // Điểm (x,y) được vẽ , đặt một chấm điểm ra giấy
  }
```

3. Tô miền

Để tô mầu cho một miền nào đó trên màn hình ta dùng hàm

```
void floodfill(int x, int y, int border);
```

ở đây:

(x,y) là tọa độ của một điểm nào đó gọi là điểm gieo.

tham số border chứa mã của một mầu.

Sự hoạt động của hàm floodfill phụ thuộc vào giá trị của x,y, border và trạng thái màn hình.

a) Khi trên màn hình có một đường (cong hoặc gấp khúc) khép kín mà mã mầu của nó bằng giá trị của border thì:

+ Miền giới hạn bởi đường kín sẽ được tô mầu nếu điểm gieo (x,y) nằm bên trong miền này.

+ Nếu điểm gieo (x,y) nằm bên ngoài thì phần màn hình bên ngoài miền đóng nói trên được tô màu.

b) Khi trên màn hình không có một đường nào như vậy, thì cả màn hình được tô màu.

Ví dụ 1. Chương trình dưới đây sẽ vẽ một đường tròn đỏ trên màn hình xanh. Tọa độ (x,y) của điểm gieo được nạp vào từ bàn phím. Tùy thuộc vào giá trị cụ thể của x,y, chương trình sẽ tô mầu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include <graphics.h>
#include <stdio.h>
main()
  {
    int mh=0, mode=0, x, y;
    initgraph(&mh, &mode, "");
    if (graphresult!= grOk) exit(1);
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(11,YELLOW);
    circle(320,100,50);
    moveto(1,150);
    outtext(" Toa do diem gieo x,y ");
    scanf("%d%d",&x,&y); flooddfill(x,y,RED);
  }
```


Ví dụ 2. Minh họa cách dùng hàm Putpixel và hàm getpixel để vẽ các điểm ảnh và sau đó xóa các điểm ảnh. Muốn kết thúc chương trình bấm ESC.

```
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
int seed = 1962; // Nhân cho bộ tạo số ngẫu nhiên
int numpts = 2000; // Vẽ 2000 chấm điểm
int ESC = 27;
void putpixelplay(void);
main()
  {
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    if (graphresult()!= grOk)
      exit(1);
    putpixelplay();
    closegraph();
  }
void putpixelplay(void)
  {
    int i,x,y,color,xmax,ymax,maxcolor,ch;
    struct viewporttype v;
    getviewsettings(&v);
    xmax=(v.right - v.left -1); ymax=(v.bottom - v.top -1);
    maxcolor=getmaxcolor();
    while (!kbhit())
      {
        //Vẽ các chấm điểm một cách ngẫu nhiên
```

```
    srand(seed);
    i=0;
    while(i<=numpts)
      {
       ++i;
       x=random(xmax)+1;y=random(ymax)+1;
       color=random(maxcolor);
       putpixel(x,y,color);
      }
    // Xóa các điểm ảnh
    srand(seed);
    i=0;
    while(i<=numpts)
      {
       ++i;
       x= random(xmax) + 1; y= random(ymax) + 1;
       color=random(maxcolor);
       putpixel(x,y,0);
      }
    if(kbhit())
      {
       ch=getch();
       if (ch==ESC) break;
      }
   }
} // Kết thúc hàm putpixelplay
```


§ 9. XỬ LÝ VĂN BẢN TRÊN MÀN HÌNH ĐỒ HOẠ

1. Hiển thị văn bản trên màn hình đồ hoạ

Hàm

```
void outtext (char *s);
```

sẽ hiện chuỗi ký tự (do s trỏ tới) tại vị trí hiện tại của con trỏ.

Hàm

```
void outtextxy(int x,int y,char *s);
```

sẽ hiện chuỗi ký tự (do s trỏ tới) tại ví trí (x,y).

Ví dụ 1: Hai cách sau đây sẽ cho cùng kết quả

```
outtextxy (100,100," chao ban ");
```

và

```
moveto (100,100);
outtext (" chao ban ");
```

Chú ý: Trong mốt đồ họa vẫn cho phép dùng hàm nhập dữ liệu scanf và các hàm bắt phím getch, kbhit.

2. Fonts

Như đã nói ở trên: Các Fonts nằm trong các tệp tin .CHR trên đĩa. Các Font này cho các kích thước và kiểu chữ khác nhau sẽ hiện thị trên màn hình đồ hoạ bằng outtext hay outtextxy. Để chọn và nạp Font chúng ta dùng hàm:

```
void settextstyle(int font,int direction,int charsize);
```

(Chú ý: hàm chỉ có tác dụng nếu tồn tại các tệp .CHR)

Với direction là một trong hai hằng số:

```
HORIZ_DIR = 0
VERT_DIR = 1
```

Nếu direction là HORIZ_DIR, văn bản sẽ hiển thị theo hướng nằm ngang từ trái sang phải. Nếu direction là VERT_DIR, văn bản sẽ hiển thị theo chiều đứng từ dưới lên trên.

Đối charsize là hệ số phóng to ký tự và có giá trị trong khoảng từ 1 đến 10.

- Nếu charsize = 1, ký tự được thể hiện trong hình chữ nhật 8*8 pixel.

- Nếu charsize = 2, ký tự được thể hiện trong hình chữ nhật 16*16 pixel.

. . .

- Nếu charsize = 10, ký tự được thể hiện trong hình chữ nhật 80*80 pixel.

Cuối cùng là tham số font để chọn kiểu chữ và nhận một trong các hằng sau:

DEFAULT_FONT = 0

TRIPLEX_FONT = 1

SMALL_FONT = 2

SANS_SERIF_FONT = 3

GOTHIC_FONT = 4

Các giá trị do settextstyle thiết lập sẽ dữ nguyên cho đến khi gọi một settextstyle mới.

Ví dụ 2:

```
settextstyle (3,VERT_DIR,2);
outtextxy (50,50," HELLO ");
```

3. Vị trí hiển thị

Hàm settextjustify cho phép ấn định nơi hiển thị văn bản của outtext theo quan hệ với vị trí hiện tại của con chạy hay của outtextxy theo quan hệ với toạ độ (x,y).

Hàm này có dạng

```
void settextjustify(int horiz, int vert);
```

Tham số horiz có thể là một trong các hằng số sau:

LEFT_TEXT = 0 (Văn bản xuất hiện bên phải con chạy)

CENTER_TEXT = 1 (Chỉnh tâm văn bản theo vị trí con chạy)

RIGHT_TEXT = 2 (Văn bản xuất hiện bên trái con chạy)

Tham số Vert có thể là một trong các hằng số sau:

BOTTOM_TEXT = 0 (Văn bản xuất hiện phía trên con chạy)

CENTER_TEXT = 1 (Chỉnh tâm văn bản theo vị trí con chạy)

TOP_TEXT = 2 (Văn bản xuất hiện phía dưới con chạy)

Ví dụ 3:

```
settextjustify(1,1);
outtextxy(100,100,"ABC");
```

Kết quả là điểm (100,100) sẽ nằm giữa chữ B.

4. Bề rộng và bề cao của văn bản

Hàm

```
void textheight (char *s);
```

trả về chiều cao (theo pixel) của chuỗi do s trỏ tới. Ví dụ nếu ký tự có kích thước 8*8 thì

```
textheight ("H") = 8
```

Ví dụ 4: Đoạn chương trình dưới đây sẽ cho hiện 5 dòng chữ.

```
#include <graphics.h>
main()
  {
    int mh=0,mode=0,y,size;
    initgraph(&mh,&mode,"");
    y=10;
    settextjustify(0,0);
    for (size=1; size<=5; ++size)
      {
        settextstyle(0,0,size);
        outtextxy(0,y,"GRAPHICS");
        y += textheight("GRAPHICS") + 10;
      }
```

```
    getch();
    closegraph();
   }
```

Hàm

```
    void textwidth(char *s);
```

sẽ dựa vào chiều dài của chuỗi, kích thước Font chữ, hệ số khuyếch đại chữ để trả về bề rộng (theo pixel) của chuỗi do s trỏ tới.

Ví dụ 5: Trong chương trình dưới đây sẽ lập các hàm vào ra trên màn hình đồ hoạ.

```
#include <graphics.h>
#include <conio.h>
#define Enter 13
#define Lmargin 10
void text_write(int *x,int *y,char *s);
void text_writeln(int *x,int *y,char *s);
void text_read(int *x,int *y,char *s);
void text_write(int *x,int *y,char *s)
  {
   outtextxy(*x,*y,s); *x += textwidth(s);
  }
void text_writeln(int *x,int *y,char *s)
  {
   outtextxy(*x,*y,s);
   *x=Lmargin;
   *y += textheight(s)+5;
  }
void text_read(int *x,int *y,char *s)
  {
   int i=0; char ch[2];
```



```
   ch[1]=0;
   while(1)
    {
     ch[0]=getch();
     if(ch[0]==Enter) break;
     text_write(x,y,ch);
     s[i]=ch[0]; ++i;
    }
   s[i]=0;
  }
main()
 {
   int mh=0,mode=0,x,y,xmax,ymax;
   char name[25];
   initgraph(&mh,&mode,"");
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
   x=Lmargin; y=100;
   text_write (&x,&y,"cho ten cua ban: ");
   text_read (&x,&y,name);
   text_writeln (&x,&y,"" );
   text_write(&x,&y,"chao ban ");
   text_write(&x,&y,name);
   getch();
   closegraph();
 }
```

§ 10. CẮT HÌNH, DÁN HÌNH VÀ TẠO ẢNH CHUYỂN ĐỘNG

1. Hàm

unsigned imagesize(int x1,int y1,int x2,int y2)

trả về số byte cần thiết để lưu trữ ảnh trong phạm vi hình chữ nhật (x1,y1,x2,y2).

2. Hàm

#include <alloc.h>

void *malloc(unsigned n);

trả về con trỏ trỏ tới một vùng nhớ n byte mới được cấp phát.

3. Hàm

void getimage(int x1,int y1,int x2,int y2,void *bitmap);

sẽ chép các điểm ảnh của hình chữ nhật (x1,y1,x2,y2) và các thông tin về bề rộng, cao của hình chữ nhật vào vùng nhớ do bitmap trỏ tới. Vùng nhớ và biến bitmap cho bởi hàm malloc. Độ lớn của vùng nhớ được xác định bằng hàm imagesize.

4. Hàm

void putimage(int x,int y,void *bitmap,int copymode);

dùng để sao ảnh lưu trong vùng nhớ bitmap ra màn hình tại vị trí (x,y). Tham số copymode xác định kiểu sao chép ảnh, nó có thể nhận các giá trị sau:

COPY_PUT = 0	Sao chép nguyên xi.
XOR_PUT = 1	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép XOR
OR_PUT = 2	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép OR
AND_PUT = 3	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép AND
NOT_PUT = 4	ảnh xuất hiện trên màn hình theo dạng đảo ngược (phép NOT) với ảnh trong bitmap.

Nhận xét: Nếu dùng mode XOR_PUT để chép hình, rồi lặp lại đúng câu lệnh đó thì hình sẽ bị xoá và màn hình trở lại như cũ. Kỹ thuật này dùng để tạo lên các hình ảnh chuyển động.

Ví dụ 1: Chương trình dưới đây minh hoạ cách dùng imagesize, malloc, getimage và putimage.

```
#include <alloc.h>
#include <graphics.h>
main()
 {
  int mh=0,mode=0;
  char *p;
  unsigend size;
  initgraph (&mh,&mode,"");
  bar(0,0,getmaxx(),getmaxy());
  size = imagesize(10,20,30,40);
  p=(char*)malloc(size); // p trỏ tới vùng nhớ size byte
                         // mới được cấp phát
  getimage (10,20,30,40,p);
  getch();
  cleardevice();
  putimage (100,100,p,COPY_PUT);
  getch();
  closegraph();
 }
```

5. Tảo ảnh di động

Nguyên tắc tạo ảnh di động giống như phim hoạt hình:

- Vẽ một hình (trong chuỗi hình mô tả chuyển động)
- Delay
- Xoá hình đó
- Vẽ hình kế theo
- Delay

. . .

A) Vẽ hình

Cách 1: Vẽ lại một ảnh nhưng tại các vị trí khác nhau.

Cách 2: Lưu ảnh vào một vùng nhớ rối đưa ảnh ra màn hình tại các vị trí khác nhau.

B) Xóa ảnh

Cách 1: Dùng hàm cleardevice

Cách 2: Dùng hàm putimage (mode XOR_PUT) để xếp chồng lên ảnh cần xoá.

Cách 3: Lưu trạng thái màn hình vào một chỗ nào đó. Vẽ một hình ảnh. Đưa trạng thái cũ màn hình ra xếp đè lên ảnh vừa vẽ.

Kỹ thuật tạo ảnh di động được minh hoạ trong các chương trình của §11.

§ 11. MỘT SỐ CHƯƠNG TRÌNH ĐỒ HOẠ

Chương trình 1: Đầu tiên vẽ bầu trời đầu sao. Sau đó từng chùm pháo hoa được bắn lên bầu trời. Khi bấm phím Enter thì việc bắn pháo hoa kết thúc, ta nhận lại bầu trời đầy sao. Bấm tiếp Enter thì kết thúc chương trình.

```
// Bắn pháo hoa trên bầu trời đầy sao
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
main()
  {
    int x[101],y[101];
    int mh=0,mode=0,i,n;
    char *p[101];
    initgraph(&mh,&mode,"");
```



```
if (graphresult()!=0) exit(1);
setcolor(RED);
// Vẽ bầu trời đầy sao
for (i=1;i<=1000;++i)
  {
    putpixel(random(getmaxx()),
    random(getmaxy()),random(getmaxcolor()));
  }
// Lưu hiện trạng 100 hình chữ nhật trên màn hình để khôi phục
for (i=1;i<=100;++i)
  {
    x[i]=random(getmaxx())-10;
    y[i]=random(getmaxy())-10;
    if (x[i]<0) x[i]=0;
    if (y[i]<0) y[i]=0;
    n=imagesize(x[i],y[i],x[i]+10,y[i]+10);
    p[i]=(char*)malloc(n);
    getimage(x[i],y[i],x[i]+10,y[i]+10,p[i]);
  }
// Chu trình bắn pháo hoa
do
  {
    // Đưa 100 quả pháo lên màn hình tại các vị trí quy định
    for (i=1;i<=100;++i)
      {
        setfillstyle(SOLID_FILL,i%15+1);
        pieslice(x[i]+5,y[i]+5,0,360,5);
      }
    delay(500);
```

```
      //Xoá chùm pháo hoa vừa bắn bằng cách khôi phục màn hình
      for (i=100;i>=1;--i)
        putimage(x[i],y[i],p[i],COPY_PUT);
      delay(500);
     } while(!kbhit());
   getch();
   getch();
   closegraph();
  }
```

Chương trình 2: Vẽ đồng hồ có 3 kim giờ, phút và giây. Đồng hồ chạy đúng theo giờ hệ thống. Muốn kết thúc chương trình bấm Enter.

```
// Đồng hồ
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
// Hàm kẻ đoạn thẳng từ tâm đồng hồ theo độ, chiều dài,
// độ dầy và mầu
void ke(int ddo, unsigned dai, unsigned day,unsigned mau);
// Kẻ kim giây khi biết số giây
void ke_giay(unsigned giay);
// Kẻ kim phút khi biết số phút
void ke_phut(unsigned phut);
// Kẻ kim giờ khi biết số giờ
void ke_gio(unsigned gio, unsigned phut);
void chay_kim_giay(void); void chay_kim_phut(void);
void chay_kim_gio(void);
int x0,y0,rgio,rphut,rgiay,mgio,mphut,mgiay;
unsigned phutgioht,gioht,phutht,giayht;
```



```
void ke(int ddo, unsigned dai, unsigned day,unsigned mau)
  {
   unsigned x,y; float goc;
   while (ddo>=360) ddo=ddo-360;
   goc=(M_PI/180)*ddo;
   x=x0+ (int)(dai*cos(goc)+0.5);
   y=y0- (int)(dai*sin(goc)+0.5);
   setcolor(mau); setlinestyle(0,0,day);
   line(x0,y0,x,y);
  }
// Hàm ke kim giay
void ke_giay(unsigned giay)
  {
   int ddo;
   ddo = (90 - 6*giay);
   ke(ddo,rgiay,1,mgiay);
  }
// Hàm ke kim phut
void ke_phut(unsigned phut)
  {
   int ddo;
   ddo= (90-6*phut);
   ke(ddo,rphut,3,mphut);
  }
// Hàm ke kim gio
void ke_gio(unsigned gio, unsigned phut)
  {
   int ddo;
   ddo = 360 + 90 - 30*(gio%12) - (phut+1)/2;
   ke(ddo,rgio,3,mgio);
  }
```

```
// Hàm chỉnh giây hiện tại và làm chuyển động kim giây
void chay_kim_giay(void)
  {
    unsigned giay; struct time t;
    gettime(&t);
    giay=t.ti_sec;
    if (giay!=giayht)
      {
        ke_giay(giayht);
        giayht=giay;
        ke_giay(giayht);
      }
  }
// Hàm chỉnh phút hiện tại và làm chuyển động kim phút
void chay_kim_phut(void)
  {
    unsigned phut;
    struct time t;
    gettime(&t);
    phut=t.ti_min;
    if (phut!=phutht)
      {
        ke_phut(phutht);
        phutht=phut;
        ke_phut(phutht);
      }
  }
```



```
// Hàm chỉnh giờ phút hiện tại và làm chuyển động kim giờ
void chay_kim_gio(void)
  {
    unsigned h,gio,phut,sophut,sophutht;
    struct time t;
    gettime(&t);
    gio=t.ti_hour; phut=t.ti_min;
    sophut = gio*60+phut;
    sophutht = gioht*60+phutgioht;
    if ( sophut<sophutht) sophut=sophut+ 12*60;
    h=sophut-sophutht;
    if (h>=12)
      {
        ke_gio(gioht,phutgioht);
        phutgioht=phut;
        gioht=gio;
        ke_gio(gioht,phutgioht);
      }
  }
main()
  {
    struct time t;
    char *dso[]={"", "12", "1", "2", "3", "4", "5", "6", "7", "8", "9",
                 "10", "11"};
    int i,mh=0,mode=0,r,x,y;
    float goc;
    initgraph(&mh,&mode,"");
    x0=(getmaxx()/2)-1; y0=(getmaxy()/2)-1;
    r=y0-2;
```

```
rgiay = r-10; rphut=r-50; rgio=r-90;
mgiay= BROWN; mphut=RED; // mgio:=magenta;
mgio=YELLOW;
// Vẽ chu vi đồng hồ
setcolor(BLUE); setlinestyle(0,0,3); circle(x0,y0,r);
setfillstyle(1,YELLOW);
floodfill(0,0,BLUE);
setfillstyle(1,WHITE); floodfill(x0,y0,BLUE);
setlinestyle(0,0,1);
circle(x0,y0,10);
setfillstyle(1,GREEN); floodfill(x0,y0,BLUE);
settextjustify(1,1); setcolor(MAGENTA);
outtextxy(x0,y0+120,"IBM-JIMIKO");
// Ghi chữ số
settextstyle(3,0,3); settextjustify(1,1); setcolor(BLUE);
for (i=1;i<=12;++i)
  {
    goc=(2*M_PI+M_PI/2) - (i-1)*(M_PI/6);
    x = x0+ (int)(rphut*cos(goc)+0.5);
    y = y0- (int)(rphut*sin(goc)+0.5);
    outtextxy(x,y,dso[i]);
  }
// Xác định thời điểm đầu
gettime(&t);
gioht=t.ti_hour; phutht=t.ti_min; giayht=t.ti_sec;
phutgioht=phutht;
setwritemode(XOR_PUT);
// Ve kim gio,phut,giay
ke_gio(gioht,phutgioht);
```



```
    ke_phut(phutht);
    ke_giay(giayht);
    // Làm chuyển động các kim
    do
      {
        chay_kim_giay(); chay_kim_phut();
        chay_kim_gio();
      }
    while(!kbhit());
    closegraph();
  }
```

Chương trình 3: Vẽ một con tầu vũ trụ bay trên bầu trời đầy sao theo quỹ đạo ellipse. Trong khi tầu chuyển động thì các ngôi sao thay nhau nhấp nháy

```
// Tầu vũ trụ chuyển động trên bầu trời đầy sao nhấp nháy
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <math.h>
// Khai báo các hàm trong chương trình
void tau_cd(void); // tầu chuyển động
void nhap_nhay_bt(void); // nhấp nháy bầu trời
void main(void); // hàm main
// Khai báo các biến mảng ngoài
int a,b,x,y,x0,y0,mh=0,mode=0,n,i;
float goc,xt,yt;
char *p;
```

```
int xx[1001],yy[1001];
// Hàm main
void main(void)
  {
    initgraph(&mh,&mode,"");
    if (graphresult()!=0) exit(1);
    // Vẽ tầu vũ trụ
    setcolor(RED);
    ellipse(100,50,0,360,20,8);
    ellipse (100,46,190,357,20,6);
    line(107,44,110,38);
    circle(110,38,2);
    line(93,44,90,38);
    circle(90,38,2);
    setfillstyle(SOLID_FILL,BLUE);
    floodfill(101,54,RED);
    setfillstyle(SOLID_FILL,MAGENTA);
    floodfill(94,45,RED);
    // Lưu ảnh của tầu vũ trụ vào bộ nhớ
    n=imagesize(79,36,121,59);
    p=(char*)malloc(n);
    getimage(79,36,121,59,p);
    // Vẽ bầu trời đầy sao và lưu vị trí của chúng
    // vào các mảng xx, yy để phục vụ hàm nhap_nhay_bt
    cleardevice();
    for (i=1;i<=1000;++i)
      {
        xx[i]=random(getmaxx()); yy[i]=random(getmaxy());
        putpixel(xx[i],yy[i],random(getmaxcolor()));
      }
```



```
    // Xác định giá trị ban đầu cho các biến
    // dùng để điều khiển chuyển động tầu
    goc= 2*M_PI + M_PI/2;
    x0= (getmaxx() - 42)/2;
    y0= (getmaxy() - 25)/2;
    a=x0; b=y0;
    // chu trình tầu vũ trụ chuyển động và các ngôi sao nhấp nháy
    do
      {
        tau_cd();
        nhap_nhay_bt();
      } while(!kbhit());
    getch();
    closegraph();
  }
void tau_cd(void)
  {
    xt=a*cos(goc)+x0;
    yt=-b*sin(goc)+y0;
    x=(int)(xt+0.5); y=(int)(yt+0.5);
    // Đặt tầu vũ trụ lên màn hình
    putimage(x,y,p,XOR_PUT);
    delay(500);
    // Xóa
    putimage(x,y,p,XOR_PUT);
    // Thay đổi góc để làm cho tầu chuyển động
    goc -= M_PI/30;
    if (goc<M_PI/2) goc=2*M_PI+M_PI/2;
  }
```

```
void nhap_nhay_bt(void)
  {
    static i=1; // Lệnh này thực hiện một lần khi dịch
    int j;
    // Cho nhấp nháy bằng cách đổi mầu 50 ngôi sao
    for (j=1;j<=50;++j)
      {
        putpixel(xx[i],yy[i],random(getmaxcolor()));
        ++i;
        if (i>1000) i=1;
      }
  }
```

§ 12. IN ẢNH TỪ MÀN HÌNH ĐỒ HOẠ

Hàm in_anh dưới đây sẽ in ảnh trong miền chữ nhật (xt, yt, xd, yd) của màn hình đồ hoạ ra giấy trên các máy in LQ1070, LQ1170 và FX1050.

```
void in_anh(int dd,int xt,int yt,int xd,int yd);
```

Tham số dd là độ đậm của nét in. Thực chất dd là số lần in lại. Bình thường chon dd=1. Nếu muốn in rõ hơn ta chọn dd bằng 2 hay 3.

Trong hàm in_anh có dùng hàm tao_mau, nó được mô tả như sau:

```
int tao_mau(int k,int x,int y);
```

Hàm này sẽ dò trên k chấm điểm theo chiều dọc bắt đầu từ toạ độ (x,y) trên màn hình để biết xem chấm điểm nào đã tô mầu. Hàm sẽ trả về một giá trị nguyên tạo bởi các bit 1 (ứng với điểm đã tô mầu) và 0 (ứng với điểm chưa tô mầu).

Hàm in_anh sẽ dùng hàm tao_mau để duyệt trên miền chữ nhật (xt,yt,xd,yd). Mỗi lần duyệt sẽ nhận được một mẫu các chấm điểm (giá trị nguyên) và mẫu này được in ra giấy.

Dưới đây là nội dung của 2 hàm nói trên.

```
// in ảnh
#include "stdio.h"
#include "graphics.h"
int tao_mau(int k,int x,int y);
void in_anh(int dd,int xt,int yt,int xd,int yd);
int tao_mau(int k,int x,int y)
  {
    int c=0,i;
    for (i=0;i<k;++i)
      if (getpixel(x,y+i)) c =c|(128>>i);
    return c;
  }
void in_anh(int dd,int xt,int yt,int xd,int yd)
  {
    //dd - so lan in lai mot dong
    char c,ch1;
    int scot,m,mm,k,dong,cot,i,j,n1,n2;
    dong=(yd-yt+1)/6; mm=(yd-yt+1) % 6;
    cot=xd-xt+1;
    for (i=0;i<=dong;++i)
      {
        if (i<dong) m=6; else m=mm;
          if (m>0)
            {
              scot=0;
              for (j=0;j < cot;++j)
                if (tao_mau(m,xt+j,yt+i*6)) scot=j+1;
```

```
            if (scot)
              {
                n1=scot % 256; n2= scot/256;
                for (k=0;k<dd;++k)
                  {
                    fprintf(stdprn,"%c%c%c%c%c%c",13,27,'*',
                                                    0,n1,n2); //LQ
                    for (j=0;j < scot;++j)
                      {
                        if (kbhit())//bat phim
                          {
                            if ((ch1=getch())==0) getch();
                            if (ch1==27) goto ket;
                          }
                        c=tao_mau(m,xt+j,yt+i*6);
                        fprintf(stdprn,"%c",c);
                      }
                  }
              }
            fprintf(stdprn,"%c%c%c",27,'A',m);
            fprintf(stdprn,"\n");
        }
      }
     ket: fprintf(stdprn,"%c%c",27,'@');
}
```