

Міністерство освіти і науки України
Національний університет “Львівська Політехніка”



Лабораторна робота №20

Виконав:
Студент групи АП-11
Білий Анатолій Іванович

Прийняв:
Чайковський І.Б.

Тема роботи: Дослідження графічного режиму роботи мови програмування С.

Мета роботи: Дослідження основних принципів відображення графічної інформації на екрані дисплея.

Попередні відомості

Для оформлення діалогу користувача з комп'ютером (програмою) потрібна розвинена система функцій управління роботою екрану. Пакет функцій управління екраном ділиться на дві частини. Перша підтримує текстовий режим (text mode) роботи. У текстовому режимі екран монітора умовно розбивається на окремі ділянки, частіше всього на 25 рядків по 80 символів (знакомісць). У кожне знакомісце може бути виведений один з 256 задалегідь заданих символів. Друга частина забезпечує роботу екрану в графічному режимі (graphics mode). Він призначений для виведення на екран графіків, діаграм, малюнків тощо. У цьому режимі екран монітора є безліччю точок, кожна з яких може бути одним із декількох кольорів. Кількість точок по горизонталі і вертикалі називається роздільною здатністю монітора в цьому режимі.

Ініціалізація графічного режиму.

До складу графічного пакету входять:

- заголовний файл graphics.h;
- бібліотечний файл graphics.lib;
- драйвери графічних пристроїв (*.bgi);
- шрифти (*.chr).

Керування екраном у графічному режимі здійснюється за допомогою набору функцій, прототипи яких знаходяться в заголовному файлі graphics.h. Для роботи в графічному режимі файл graphics.h має бути підключений за

допомогою директиви `#include` препроцесора мови C до всіх модулів, що використовують графічні підпрограми `#include <graphics.h>`.

Перш ніж використовувати графічні функції, необхідно перемкнути відеоадаптер у графічний режим (за замовчуванням він знаходиться в текстовому режимі). Для ініціалізації графічного режиму призначена функція `initgraph()`.

Її прототип:

```
void initgraph(int *driver, int *mode, char *path);
```

де `int *driver` - тип драйвера, що підключається, `int *mode` - режим роботи підключеного драйвера, `char*path` - місце розташування драйвера.

Функція `initgraph()` прочитує в пам'ять вказаний драйвер, встановлює відеорежим, що відповідає аргументу `mode`, і визначає маршрут до директорії, в якій знаходиться файл `*.bgi` (драйвер). Якщо маршрут не вказаний, то передбачається, що цей файл розташований в поточній директорії (зазвичай директорія `bin`).

При використанні `initgraph()` можна вказати або конкретний драйвер (наприклад, `egavga.bgi`), або задати автоматичне визначення (детектування) типу відеоадаптера і вибору відповідного драйвера вже під час виконання програми (макрос `DETECT`). Це дозволяє без зміни переносити програми на комп'ютери з іншими відеоадаптерами. Наприклад:

```
int grdrv=DETECT, grmod;  
initgraph(&grdrv,&grmod," ");
```

У наведеному прикладі мається на увазі, що файл драйвера (наприклад, `egavga.bgi`) розташований в поточній директорії. Ця функція очищає екран монітора і переводить його в графічний режим 640 X 480 * 16 (роздільна здатність монітора 640 X 480, 16 кольорів).

Щоб вийти з графічного режиму і повернутися в текстовий режим, необхідно використати функцію `void closegraph(void)`;

Функція `closegraph()` звільняє пам'ять, використовувану графічними функціями, і встановлює текстовий режим, який був до виклику функції `initgraph()`, при цьому відбувається очищення екрану.

Палітра. Найчастіше використовується графічний режим монітора, при якому підтримується роздільна здатність 640 X 480 * 16. Тут 16 – максимальна

кількість кольорів, які одночасно можуть бути присутні на зображенні. У файлі `graphics.h` визначені константи, що відповідають кольорам стандартної (використовуваної за замовчуванням) палітри.

Кольори задають числами, або англійськими назвами (великі букви, Таблиця 1).

Таблиця 1.

BLACK (0)	чорний	DARKGRAY (8)	темний сірий
BLUE (1)	синій	LIGHTBLUE (9)	яскравий синій
GREEN (2)	зелений	LIGHTGREEN (10)	яскравий зелений
CYAN (3)	блакитний	LIGHTCYAN (11)	яскравий блакитний
RED (4)	червоний	LIGHTRED (12)	червоний
MAGENTA (5)	фіолетовий	LIGHTMAGENTA (13)	фіолетовий
BROWN (6)	коричневий	YELLOW (14)	жовтий
LIGHTGRAY (7)	світлий сірий	WHITE (15)	білий

Зміна одного з кольорів стандартної палітри здійснюється функцією:

```
void setpalette(int index, int color);
```

тут `int index` - номер із стандартної палітри, а `int color` - колір в діапазоні від 0 до 63 (палітра EGA).

Налаштування палітри EGA здійснюється функцією

```
void setrgbpalette(int color, int red, int green, int blue);
```

де `red`, `green` і `blue` змінюються в діапазоні від 0 до 255. Малим значенням `red`, `green` і `blue` відповідають темні кольори, великим - яскравіші. Якщо `red`, `green` і `blue` мають однакові значення, то формується один з відтінків сірого кольору.

Нижче наведений приклад, в якому з використанням цих функцій отриманий новий набір кольорів (різні відтінки сірого кольору).

```
for (i=0; i<16; i++)  
{  
    setrgbpalette(i, i*16, i*16, i*16);  
    setpalette(i, i);} 
```

Графічний екран є масивом пікселів. Кожен піксель відповідає одній точці на екрані і може мати свій колір. Встановити колір пікселя color в точці екрану з координатами (x, y) можна за допомогою функції void putpixel(int x, int y, int color);

Також існує зворотна функція, яка визначає колір точки з координатами (x, y) unsigned getpixel(int x, int y);

Основні функції для графічних побудов:

```
setcolor (колір);  
setbkcolor (колір) колір тла.;  
putpixel (x,y,колір);  
line(x1,y1,x2,y2);  
lineto(x,y);  
bar(x1,y1,x2,y2);  
rectangle(x1,y1,x2,y2);  
circle(x,y,R);  
arc(x,y,початк. кут, кінцевий кут, радіус);  
closegraph();  
outtext(текст);  
outtextxy(x,y,текст);  
settextstyle(шрифт, напрям, розмір);  
getmaxx() – повертає розмір екрану по горизонталі;  
getmaxy() – повертає розмір екрану по вертикалі;  
getcolor() – повертає значення поточного кольору;  
getx(), gety() – повертають координати поточного пікселя.
```

Група ліній на площині утворює контурну фігуру. Типовими представниками контурних фігур можна вважати відрізок прямої лінії, дугу, коло, прямокутник, еліпс і т. д.

Окрім форми одна фігура від іншої може відрізнятися кольором лінії (контуру), її товщиною або типом. За замовчуванням у графічному режимі існують наступні налаштування: поточний колір контуру - WHITE (білий), товщина – один піксель, тип – суцільна лінія.

Змінити колір рисування можна, звернувшись до функції `setcolor()` із прототипом `void setcolor(int color);`

Інші параметри контурів встановлюються функцією:

`void setlinestyle(int linestyle, unsigned upattern, int thickness);`

тут: `linestyle` - тип лінії, а `thickness` - її товщина. Лінія може бути п'яти типів:

0 – `SOLID_LINE` (суцільна);

1 – `DOTTED_LINE` (пунктирна);

2 – `CENTER_LINE` (штрих-пунктирна);

3 – `DASHED_LINE` (штрихова);

4 – `USERBIT_LINE` (тип, що визначається користувачем).

Для відображення фігур, що найчастіше використовуються можна скористатися функціями, які вже є в стандартній графічній бібліотеці.

Розглянемо деякі з них. Функція

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

креслить на екрані пряму лінію від точки з координатами (x1, y1) до точки з координатами (x2, y2).

Функція

`void rectangle(int left, int top, int right, int bottom);`

креслить прямокутник, розташований на екрані горизонтально (вертикально) з координатою лівого верхнього кута (left, top) і правого нижнього – (right, bottom).

Для відображення кола з центром в точці (x, y) і радіусом `radius` (одиниця виміру – піксель) необхідно скористатися функцією:

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

Якщо необхідно відобразити дугу, то слід скористатися функцією:

```
void arc(int x, int y, int stangle, int endangle, int radius);
```

яка викреслює дугу з центром в (x, y) і радіусом radius. Параметри stangle і endangle задають кругові координати початкової та кінцевої точок. Кут вимірюється в градусах і відлічується проти годинникової стрілки, де 0 градусів відповідає трьом годинам на циферблаті. Якщо stangle дорівнює 0, а endangle дорівнює 360, функція arc() малює повне коло.

Крім того, існують функції drawpoly() для рисування багатокутника, ellipse() – для рисування еліпса і ряд інших.

Площинні фігури є фрагментами площини екрану, обмежені замкнутим контуром. Їх можна отримати з контурних шляхом зафарбовування області всередині або поза замкнутої лінії, що утворює контур. Лінія при цьому має бути суцільною. Цю операцію можна виконати за допомогою функції:

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

тут x і y – координати точки, розташованої усередині або поза контуром відповідно (для розфарбовування тієї або іншої області), border – колір лінії, що утворює контур. При цьому колір усього контуру має бути однаковим. Зафарбування здійснюється кольором color і за шаблоном pattern, які встановлюються функцією:

```
void setfillstyle(int pattern, int color);
```

Для площинних фігур, що найчастіше зустрічаються на практиці, в графічній бібліотеці так само існують вже готові функції. Функція

```
void bar(int left, int top, int right, int bottom);
```

рисує зафарбований прямокутник. Прямокутник зафарбовується поточним кольором і з використанням поточного шаблону заповнення. Верхній лівий і нижній правий кути прямокутника задані параметрами (left, top) і (right, bottom) відповідно. Координати даються в пікселях.

Функція:

```
void fillellipse(int x, int y, int rx, int ry);
```

рисує еліпс із центром в точці (x, y), горизонтальній і вертикальній вісями rx і ry відповідно, і зафарбовує його поточним кольором, який використовує текучий шаблон.

Функція:

`void fillepoly(int numpoints, int *polipoints);`

рисує контур багатокутника, що має numpoints точок, а потім зафарбовує його. Polipoints – вказівник на послідовність з (numpoints*2) цілих чисел. Кожна пара чисел (x, y) є координатами вершини багатокутника.

Функція:

`void pieslice(int x, int y, int stangle, int endangle, int radius);`

рисує і зафарбовує сектор круга з центром в точці (x, y) і радіусом radius. Сектор рисується від кута stangle до кута endangle. Кут вимірюється в градусах і відлічується проти годинникової стрілки, де 0 градусів відповідає трьом годинам на циферблаті.

Виведення тексту в графічному режимі можна здійснити з використанням функції outtextxy().

Функція:

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

виводить поверх існуючого на екрані зображення рядок тексту textstring починаючи з позиції (x, y) (координата лівого верхнього кута першого символу рядка). Сформувавши рядок textstring необхідно заздалегідь. Для цього зручно використати функцію sprintf(). Текстова інформація відображається на екрані з урахуванням параметрів:

колір, тип шрифту, розмір шрифту і напрям. Ці параметри налаштовуються функціями:

`void setcolor(int color);`

і

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

тут color – колір тексту, font – тип шрифту, direction – напрям виведення текстової інформації (0 – горизонтальне, 1 – вертикальне) і charsize – множник, який впливає на розмір символів, що виводяться на екран.

Розмір символів (вертикальний і горизонтальний) визначається як добуток стандартного розміру (8 X 8 пікселів) на параметр charsize, тобто якщо значення charsize дорівнюватиме 3, то кожен символ, що відображається на екрані, буде вписаний в квадрат 24 * 24 пікселі.

Параметр `font`, який задає стиль шрифту, підключає до програми файли з розширенням `*.chr` (нестандартні «шрифти»), тому необхідно зробити ці файли доступними (найпростіше скопіювати їх в поточну директорію).

Для виведення тексту на екран в графічному режимі можна використати і функції для текстового режиму (наприклад, `printf()`), проте вони мають ряд недоліків. Наприклад, при використанні функції `printf()` для виведення тексту на якому-небудь кольоровому фоні позаду напису з'явиться її «фон» (чорний прямокутник, рівний довжині тексту, що виводиться). Також відсутня можливість зміни зовнішнього вигляду тексту (розміру шрифту, стилю і так далі).

Робота з частинами екрану. Досить часто при роботі з графікою виникає ситуація, коли фрагмент зображення необхідно перемістити в інше місце на екрані. Вирішити це завдання можна декількома способами. Найбільш ефективний з них припускає використання функцій `getimage()` і `putimage()`.

Функція

```
void getimage(int left, int top, int right, int bottom, void *bitmap);
```

копіює образ з екрану в оперативну пам'ять.

`Left`, `top`, `right` і `bottom` визначають область екрану прямокутної форми, яка буде скопійована. `Bitmap` вказує на область в пам'яті, куди записується бітовий образ. Перші два слова цього образу задають ширину і висоту прямокутника, інші, що залишилися визначають сам образ.

Функція

```
void far putimage(int left, int top, void far *bitmap, int op);
```

поміщає бітовий образ, раніше збережений за допомогою `getimage()`, назад на екран. Лівий верхній кут нового місця фрагмента на екрані має координати (`left`, `top`). `Bitmap` – вказівник на область пам'яті, де зберігався образ. Параметр `op` визначає колір для кожної точки екрану з урахуванням уже існуючого в цьому місці зображення та образу, що зберігається в пам'яті. Перелік значень `putimage_op` (яке визначене в `graphics.h`) дає назви наступним операціям:

0 – `COPY_PUT` (копія):

1 – `XOR_PUT` (виключне «або»);

2 – `OR_PUT` («логічне або»);

3 – AND_PUT («логічне і»;

4 – NOT_PUT (копія джерела з інверсією).

Іншими словами, COPY_PUT копіює бітовий образ джерела на екран, XOR_PUT виконує логічну операцію «виключне або» над образом, що вже знаходиться на екрані, і т. д.

Завдання визначення об'єму оперативної пам'яті (у байтах), необхідного для зберігання бітового образу фрагмента екрану, обмеженого розмірами left, top, right і bottom (ліворуч, згори, справа і знизу відповідно), можна вирішити за допомогою функції:

```
unsigned imagesize(int left, int top, int right, int bottom);
```

Приведені вище функції можна використати для створення руху образу. В результаті виконання наступної програми по екрану зліва направо рухатиметься червоний круг.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int main(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    int xmax, ymax;
    int x, y, i, k, size;
    void *buf;
    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
    /* read result of initialization */
    errorcode = graphresult();
    /* an error occurred */
    if (errorcode != grOk)
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
```

```

printf("Press any key to halt :");
getch();
exit(1);
}
setcolor(BLACK);
setfillstyle(1, RED);
fillellipse(21,240,20,20);
buf=malloc(size);
getimage(1,220,41,260, buf);
setfillstyle(1, BLACK);
for(i=1;i<=620;i++)
{
bar(i - 1,220,40+1,260);
putimage(i, 220, buf, COPY_PUT);
delay(10);
}
free(buf);
}

```

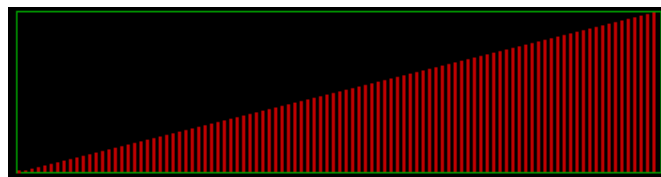
У наведеному прикладі використана функція:

```
void delay(unsigned milliseconds);
```

яка затримує виконання програми на інтервал milliseconds (затримка обчислюється в мілісекундах). Ця функція визначена в заголовному файлі dos.h.

ЗАВДАННЯ

1. Нарисувати графік функції $y = \sin(x)$ шляхом табулювання її значень на ділянці $[0, 3\pi]$ з кроком 0.1. Графік може зображатися за допомогою пікселів або вертикальних ліній.



Приклад: графік лінійно зростаючих чисел

2. Розробити простий графічний редактор, що дозволяє набирати на екрані монітора довільний текст (з можливістю динамічного вибору шрифту і

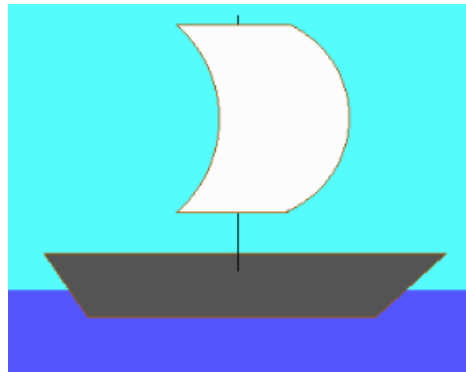
розміру символів у діалоговому режимі). Для вирішення завдання використати функцію.

3. Розробити програму для виведення в графічному режимі зображення об'єкта (кулі), що рухається по діагоналі екрану.

Для імітації руху зображення об'єкта на екрані необхідно виконати такий алгоритм: 1. Нарисувати об'єкт у заданій точці. 2. Витерти об'єкт, замалювавши його кольором тла. 3. Змінити координати об'єкта. 4. Перейти до пункту 1.

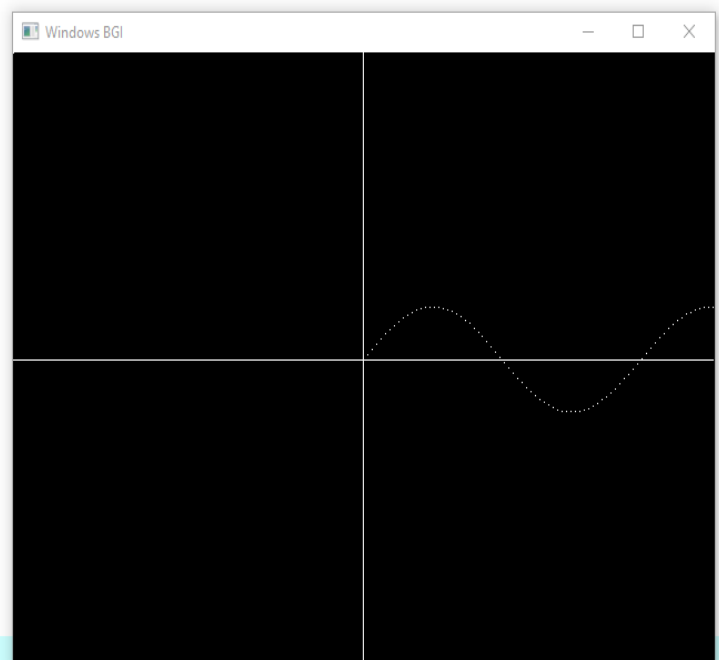
4. Розробити програму для виведення на екран у графічному режимі зображення кулі, що обертається.

5. Здійснити рух графічного об'єкта зліва направо по екрану. Для вирішення завдання використати функцію.



Завдання 1

```
1  #include <graphics.h>
2  #include <math.h>
3  #include <conio.h>
4
5  #define PI 3.14159265
6
7  int main() {
8      int gd = DETECT, gm;
9      initgraph(&gd, &gm, "");
10
11      float x, y;
12      int midX = getmaxx() / 2;
13      int midY = getmaxy() / 2;
14
15      line(midX, 0, midX, getmaxy());
16      line(0, midY, getmaxx(), midY);
17
18      for (x = 0; x <= 3 * PI; x += 0.1) {
19          y = sin(x);
20          putpixel(midX + x * 40, midY - y * 40, WHITE);
21      }
22
23      getch();
24      closegraph();
25      return 0;
26 }
27
```



```
C:\Users\Andrey\Desktop\Graphics in Dev C++\Test Graphics.h\Test Graphics.exe
Enter text: Max
Enter coordinates (x y): 9 6
Choose font (0-10): 1
Choose text direction (0=horizontal, 1=vertical): 0
Enter font size: 20
_
```

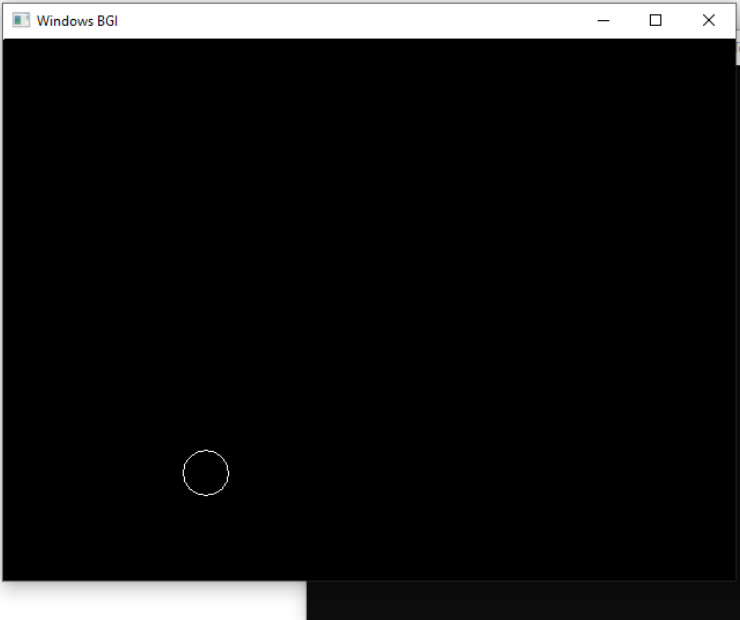
Завдання 2

```
1  #include <graphics.h>
2  #include <conio.h>
3  #include <stdlib.h>
4  void drawText() {
5      int x, y;
6      char text[100];
7      int font, direction, size;
8      printf("Enter text: ");
9      scanf("%s", text);
10     printf("Enter coordinates (x y): ");
11     scanf("%d %d", &x, &y);
12     printf("Choose font (0-10): ");
13     scanf("%d", &font);
14     printf("Choose text direction (0=horizontal, 1=vertical): ");
15     scanf("%d", &direction);
16     printf("Enter font size: ");
17     scanf("%d", &size);
18     settextstyle(font, direction, size);
19     outtextxy(x, y, text);
20 }
21 int main() {
22     int gd = DETECT, gm;
23     initgraph(&gd, &gm, "");
24     while (1) {
25         cleardevice();
26         drawText();
27         getch();
28     }
29     closegraph();
30     return 0;
31 }
32
```

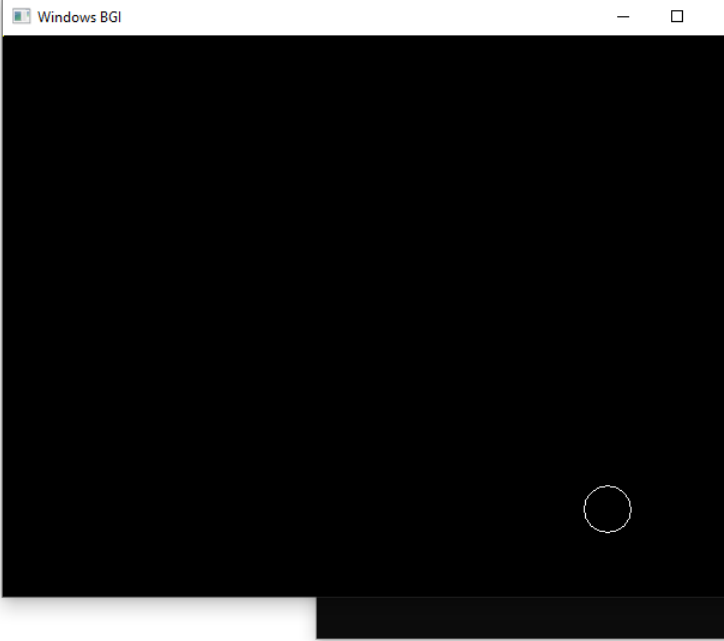


Завдання 3

```
1  #include <graphics.h>
2  #include <conio.h>
3
4  int main() {
5      int gd = DETECT, gm;
6      initgraph(&gd, &gm, "");
7
8      int x = 50, y = 50;
9      int radius = 20;
10     int dx = 5, dy = 5;
11
12     while (!kbhit()) {
13         setcolor(WHITE);
14         circle(x, y, radius);
15         delay(50);
16         setcolor(BLACK);
17         circle(x, y, radius);
18
19         x += dx;
20         y += dy;
21
22         if (x >= getmaxx() - radius || x <= radius) {
23             dx = -dx;
24         }
25         if (y >= getmaxy() - radius || y <= radius) {
26             dy = -dy;
27         }
28     }
29
30     closegraph();
31     return 0;
32 }
```



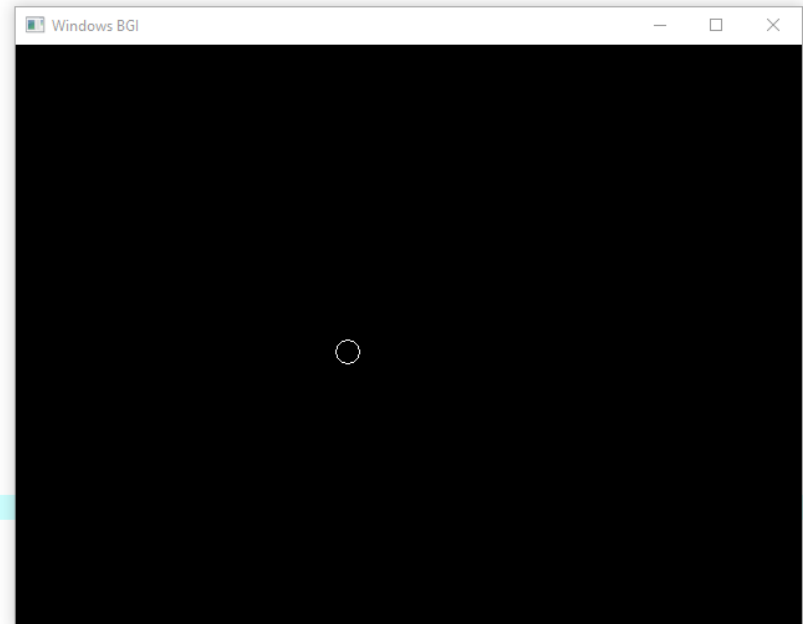
```
1  #include <graphics.h>
2  #include <conio.h>
3
4  int main() {
5      int gd = DETECT, gm;
6      initgraph(&gd, &gm, "");
7
8      int x = 50, y = 50;
9      int radius = 20;
10     int dx = 5, dy = 5;
11
12     while (!kbhit()) {
13         setcolor(WHITE);
14         circle(x, y, radius);
15         delay(50);
16         setcolor(BLACK);
17         circle(x, y, radius);
18
19         x += dx;
20         y += dy;
21
22         if (x >= getmaxx() - radius || x <= radius) {
23             dx = -dx;
24         }
25         if (y >= getmaxy() - radius || y <= radius) {
26             dy = -dy;
27         }
28     }
29
30     closegraph();
31     return 0;
32 }
```



```

1  #include <graphics.h>
2  #include <conio.h>
3  #include <math.h>
4
5  int main() {
6      int gd = DETECT, gm;
7      initgraph(&gd, &gm, "");
8
9      int x = getmaxx() / 2, y = getmaxy() / 2;
10     int radius = 50;
11     float angle = 0;
12
13     while (!kbhit()) {
14         cleardevice();
15         int circleX = x + radius * cos(angle);
16         int circleY = y + radius * sin(angle);
17         circle(circleX, circleY, 10);
18         angle += 0.05;
19         delay(50);
20     }
21
22     closegraph();
23     return 0;
24 }
25

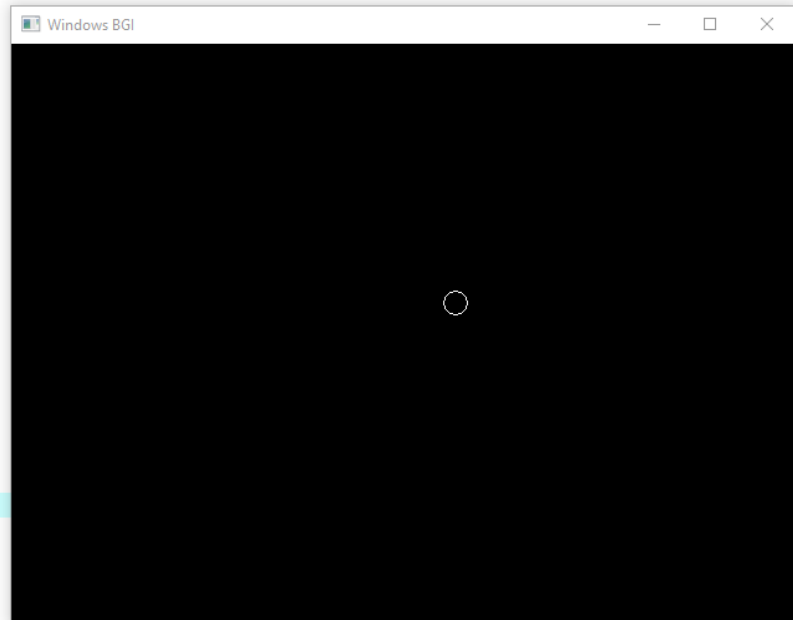
```



```

1  #include <graphics.h>
2  #include <conio.h>
3  #include <math.h>
4
5  int main() {
6      int gd = DETECT, gm;
7      initgraph(&gd, &gm, "");
8
9      int x = getmaxx() / 2, y = getmaxy() / 2;
10     int radius = 50;
11     float angle = 0;
12
13     while (!kbhit()) {
14         cleardevice();
15         int circleX = x + radius * cos(angle);
16         int circleY = y + radius * sin(angle);
17         circle(circleX, circleY, 10);
18         angle += 0.05;
19         delay(50);
20     }
21
22     closegraph();
23     return 0;
24 }
25

```



Завдання 5

```
1  #include <graphics.h>
2  #include <conio.h>
3
4  void drawShip(int x, int y) {
5      setcolor(WHITE);
6      setfillstyle(SOLID_FILL, BLUE);
7      rectangle(x, y, x + 100, y + 20);
8      floodfill(x + 1, y + 1, WHITE);
9
10     line(x + 25, y, x + 50, y - 20);
11     line(x + 50, y - 20, x + 75, y);
12     setfillstyle(SOLID_FILL, LIGHTBLUE);
13     floodfill(x + 50, y - 10, WHITE);
14
15     setcolor(WHITE);
16     rectangle(x + 80, y - 5, x + 90, y + 5);
17     setfillstyle(SOLID_FILL, RED);
18     floodfill(x + 81, y, WHITE);
19
20     setcolor(WHITE);
21     line(x + 50, y - 20, x + 50, y - 30);
22     line(x + 50, y - 30, x + 60, y - 25);
23     line(x + 60, y - 25, x + 50, y - 20);
24     setfillstyle(SOLID_FILL, GREEN);
25     floodfill(x + 51, y - 25, WHITE);
26
27     setcolor(WHITE);
28     setfillstyle(SOLID_FILL, YELLOW);
29     circle(x + 15, y + 10, 5);
30     floodfill(x + 15, y + 10, WHITE);
31     circle(x + 35, y + 10, 5);
32     floodfill(x + 35, y + 10, WHITE);
33     circle(x + 55, y + 10, 5);
34     floodfill(x + 55, y + 10, WHITE);
35 }
```



```

34     floodfill(x + 55, y + 10, WHITE);
35
36     setcolor(WHITE);
37     rectangle(x + 65, y - 10, x + 75, y + 5);
38     setfillstyle(SOLID_FILL, DARKGRAY);
39     floodfill(x + 66, y, WHITE);
40 }
41
42 int main() {
43     int gd = DETECT, gm;
44     initgraph(&gd, &gm, "");
45     int x = 0, y = getmaxy() / 2;
46     while (x < getmaxx()) {
47         cleardevice();
48         drawShip(x, y);
49         delay(50);
50         x += 5;
51     }
52
53     getch();
54     closegraph();
55     return 0;
56 }
57

```


Контрольні запитання.

1. Текстовий і графічний режими роботи в мові програмування C.

Текстовий режим

У текстовому режимі програміст взаємодіє з користувачем через консоль, використовуючи текстові команди для введення і виводу даних.

Графічний режим

Графічний режим використовує графічний інтерфейс, включаючи вікна, кнопки, малюнки та інші елементи. Для цього застосовуються додаткові бібліотеки, такі як SDL або OpenGL

2. Керування кольором і вибір палітри в мові C.

Керування кольором

Для роботи з кольорами в C зазвичай використовують графічні бібліотеки, такі як SDL або OpenGL.

Ініціалізація кольору: Визначення кольорів через структури RGB.

Встановлення кольору: Використання функцій бібліотеки для встановлення кольору об'єктів.

Палітра

Бібліотеки надають функції для роботи з палітрами, що дозволяє змінювати кольори елементів інтерфейсу.

3. Основні функції для графічного режиму роботи в мові C.

Бібліотека SDL:

Ініціалізація та завершення

SDL_Init(): Ініціалізація SDL.

SDL_Quit(): Завершення роботи з SDL.

Вікно

SDL_CreateWindow(): Створення вікна.

SDL_DestroyWindow(): Знищення вікна.

Рендерер

SDL_CreateRenderer(): Створення рендерера.

SDL_DestroyRenderer(): Знищення рендерера.

Кольори та малювання

SDL_SetRenderDrawColor(): Встановлення кольору.

SDL_RenderClear(): Очищення екрану.

SDL_RenderPresent(): Оновлення екрану.

SDL_RenderDrawLine(): Малювання лінії.

SDL_RenderDrawRect(): Малювання прямокутника.

SDL_RenderFillRect(): Заповнення прямокутника.

Обробка подій

SDL_PollEvent(): Обробка подій.

4. Принципи роботи з частинами графічного екрану в мові C.

Ініціалізація: Створення вікна та рендерера для графічного виведення.

Координатна система: Графічний екран має свою систему координат, де (0,0) - верхній лівий кут.

Області рендерингу: Використання SDL_Rect для визначення та обробки частин екрану.

Малювання: Встановлення кольору та використання функцій для малювання геометричних фігур.

Оновлення екрану: Після малювання необхідно оновити вікно для відображення змін.

Висновок: під час виконання цієї лабораторної роботи я дослідив основні принципи відображення графічної інформації на екрані дисплея.