TRYB GRAFICZNY

Dwa tryby pracy monitora:

→ **tryb tekstowy (alfanumeryczny) –** podstawową jednostką jest znak, podział ekranu na 80 kolumn × 25 wierszy × 2 bajty *{znak + atrybut}*

→ **tryb graficzny –** podstawową jednostką jest punkt (pixel),

np.	VGA High	640×480 punktów,
	VGA Medium	640×350 punktów,
	VGA Low	640×200 punktów,

W środowisku programistycznym firmy Borland (BorlandC 3.1 i Turbo Pascal) obsługa grafiki odbywała się za pomocą interfejsu BGI (*Borland Graphic Interface*) uniezależniającego programy od konkretnego sprzętu komputerowego (*różnorodność kart graficznych: Hercules, CGA, EGA, MCGA, VGA, IBM8514, SVGA, itd.*)

Funkcje i stałe interfejsu BGI zawarte są w bibliotece <g**raphics.h>**
Ta biblioteka nie jest standardową biblioteką języka C i fakt jej wykorzystania trzeba "włączyć" w opcji środowiska: *Options* | *Linker* | *Libraries* | *Graphics library* .

Demonstracja możliwości trybu graficznego w programie: **BorlandC/bgi/bgidemo.c**

Inicjacja trybu graficznego:

void **initgraph** (int* Driver, int *Mode, char* Path)

Wykorzystanie trybu graficznego wymaga wcześniejszego wykonania operacji inicjacji (*poprzez wywołanie* **initgraph)** w czasie której trzeba podać:

- identyfikator sterownika (*parametr Driver*) związany z typem karty graficznej (np. ***CGA*** *= 1,* ***MCGA*** *= 2,* ***EGA*** *= 3,* **. . .** , ***VGA*** *= 9*)
- identyfikator trybu pracy (*parametr Mode*) związany z rozdzielczością ekranu i wykorzystywaną liczbą kolorów
  (*np.* ***VGALo*** *= 0 ,* ***VGAMed*** *= 1 ,* ***VGAHi*** *= 3*)
- ścieżkę dostępu do sterownika (*parametr Path*) zawierającą informację gdzie na dysku znajduje się plik tzw. *drivera* o rozszerzeniu *.**bgi** (np.**egavga.bgi)**

Zakończenie pracy trybu graficznego dokonuje funkcja:

void **closegraph**(void)

która zwalnia pamięć przydzieloną sterownikowi, pamięć zajmowaną przez krój pisma i przywraca tryb tekstowy.

Otwarcie grafiki może się zakończyć niepowodzeniem, dlatego zaraz potem powinno nastąpić sprawdzenie stanu systemu graficznego za pomocą funkcji:

int **graphresult**(void)

która zwraca kod błędu ostatniej operacji graficznej np.

grOK	=	0
grNoInitGraph	=	–1
grNotDetected	=	–2
grFileNotFound	=	–3
		. . .

Przykładowy schemat korzystania z trybu graficznego:

```
#include <graphics.h>        // konieczne !
#include <stdlib.h>          // dla potrzeb programu demonstracyjnego
#include <conio.h>           // który wykorzystuje printf() i getch()
void main()
  {
    int driver=VGA,  mode=VGAMED, err ;
    initgraph( &driver, &mode, " \\bc31\\bgi " );
    if( ( err=graphresult() ) != grOK )    //zapamiętanie numeru błędu
      {
        printf( "\nBlad inicjacji grafiki\n" );
        printf( grapherrormsg(err) );       //wyśw. informacji o błędzie
        exit(1);
      }
    • • •
    line( 10, 20, 100, 50);   // przykładowe wyrysowanie linii na ekranie
    getch();                  // i oczekiwanie na przyciśnięcie klawisza
    • • •
    closegraph();
  }
```

Autodetekcja rodzaju karty:

W przypadku gdy nie znamy typu karty graficznej wykorzystywanej w komputerze (lub gdy chcemy aby program działał dla wszystkich kart) można zarządać automatycznego rozpoznania typu karty poprzez podanie identyfikatora **DETECT** lub poprzez rozpoznanie karty za pomocą funkcji **detectgraph(**int* driver, int* mode**)**

```
np.    • • •
       int driver = DETECT,  mode, err ;
       initgraph( &driver, &mode, " \\bc31\\bgi " );
       • • •
```

Podstawowe funkcje wykreślania linii i figur

1. PUNKTY i KOLORY:
   - void **putpixel**(int x, int y, int Kolor) // nadanie punktowi (**x,y**) koloru **Kolor**
   - int **getmaxcolor**(void) // podaje maksymalny numer koloru
   - void **setbkcolor**(int Kolor) // ustawia kolor tła
   - void **setcolor**(int Kolor) // ustawia kolor lini i konturu
2. LINIE PROSTE − ODCINKI
   - void **line**(int x1, int y1, int x2, int y2) // odcinek pomiędzy (**x1,y1**) i (**x2,y2**)
   - void **linerel**(int dx, int dy) // odcinek względem pozycji kursora
   - void **lineto**(int x, int y) // odcinek od kursora do punktu (**x,y**)
   - void **drawpoly**(int N, int tab[ ]) // linia łamana o **N** wierzchołkach, tablica // **tab** zawiera {x1,y1, x2,y2, . . . , x_N,y_N}
   - void **rectangle**(int x1, int y1, int x2, int y2) // prostokąt
   - void **setlinestyle**(int Styl, unsigned Wzor, int Grubosc) // rodzaj i grubość linii
3. LINIE KRZYWE – ŁUKI
   - void **circle**(int x, int y, int R) // okrąg o środku (**x,y**) i promieniu **R**
   - void **arc**(int x, int y, int Alfa, int Beta, int R) // łuk od kąta **Alfa** do **Beta**
   - void **ellipse**(int x, int y, int Alfa, int Beta, int Rx, int Ry) // elipsa
4. FIGURY WYPEŁNIONE
   - void **setfillstyle**(int Wzor, int Kolor) // ustawia wzór i kolor wypełnienia
   - void **setfillpattern**(char* Wzor, int Kolor) // definiuje nowy wzór wypełnienia
   - void **bar**(int x1, int y1, int x2, int y2) // prostokąt wypełniony
   - void **bar**(int x1, int y1, int x2, int y2, int Szer, int Kier) // słupek 3-wymiarowy
   - void **fillpoly**(int N, int tab[ ]) // wielokąt o **N** wierzchołkach
   - void **fillellipse**(int x, int y, int Rx, int Ry) // elipsa o promieniach **Rx** i **Ry**
   - void **pieslice**(int x, int y, int Alfa, int Beta, int R) // wycinek koła
   - void **sector**(int x, int y, int Alfa, int Beta, int Rx, int Ry) // wycinek elipsy
   - void **floodfill**(int x, int y, int Kolor) //wypełnianie obszaru ograniczonego // kolorem poczynając od (**x,y**)

Sterowanie ekranem i kursorem

1. OPERACJE NA EKRANIE LUB OKNIE
   - void **cleardevice**(void) // wyczyszczenie całego ekranu
   - void **setviewport**(int x1, int y1, int x2, int y2, int Clipp) // zdefiniowanie // okna o współrzędnych (**x1,y1**)(**x2,y2**) z ustawieniem obcinania na granicy
   - void **clearviewport**(void) // czyszczenie aktualnego okna
   - void **getviewsettings**(struct viewporttype *viewport) // odczyt parametrów okna

2. OPERACJE NA KURSORZE

```
int getmaxx( void )                 // poziomy rozmiar ekranu w pikselach
int getmaxy( void )                 // pionowy rozmiar ekranu w pikselach
int getx( void )                    // aktualna współrzędna x kursora w oknie
int gety( void )                    // aktualna współrzędna y kursora  w oknie
void moveto( int x, int y )         // przesuniecie kursora do punktu (x,y)
void moverel( int dx, int dy )      // przesuniecie kursora o wektor (dx,dy)
```

3. ZAPAMIĘTANIE RYSUNKU

```
unsigned imagesize( int x1, int y1, int x2, int y2 )      // wylicza rozmiar w bajtach
void getimage( int x1,int y1,int x2,int y2,void* buf )    // zapamiętanie rysunku
void putimage( int x1, int y1, void* buf, int Sposob ) // odtworzenie rysunku
                    // Sposob = { COPY_PUT, XOR_PUT, OR_PUT, AND_PUT }
```

4. WYBÓR STRONY GRAFICZNEJ (dotyczy tylko EGA, VGA, Hercules)

```
void setactivepage( int Strona )          // wybór strony aktywnej
void setvisualpage( int Strona )          // wybór strony wyświetlanej
```

Wykreślanie-rysowanie tekstów

W trybie graficznym teksty są wyświetlane podobnie jak wszystkie inne obiekty. Teksty mogą być bitmapowe lub kreskowe. Teksty kreskowe zapewniają dobrą jakość wizualną niezależnie od powiększenia, ale ich krój musi być załadowany z pliku. Standardowe kroje są przechowywane w tym samym katalogu co *driver* i mają rozszerzenie (∗**.chr**) .

Kroje czcionek można dołączyć do pliku (∗**.exe**) przetwarzając pliki (∗**.chr**) do postaci (∗**.obj**) za pomocą programu BINOBJ.EXE. Przetworzone kroje trzeba następnie dołączyć do pozostałych plików na etapie linkowania (zbudować projekt zawierający oprócz plików ∗**.cpp**, także w/w przetworzone pliki czcionek o rozszerzeniu ∗**.obj**)

```
void outtext( char* Text )          // wyświetlenie tekstu względem pozycji kursora
void outtextxy(int x, int y, char* Text)    // wyświetlenie względem pozycji (x,y)
```

Standardowe funkcje umożliwiają tylko wyświetlanie dany reprezentowanych w formie łańcuchów znaków. Aby wyświetlić na ekranie dane innych typów trzeba je wcześniej przetworzyć do postaci łańcuchowej np za pomocą funkcji **sprintf( )**

```
np.     float  x=123.456;       // zmienna x której wartość chcemy wyświetlić
        char bufor[ 100 ];
        sprintf( bufor, "X = %5.2f ", x );
```

```
void outtext( bufor );                  // wyświetlenie wartości x na ekranie
void settextstyle( int Krój, int Kierunek, int Rozmiar)  // zadanie parametrow tekstu
void  setusercharsize( int xMul, int xDiv, int yMul, int yDiv)  // rozciaganie tekstu
```