ABC programowania w C++

Autor: Jan Rusek
ISBN: 83-7197-558-9
Format: B5, stron: około 200

Część pierwsza książki jest rodzajem samouczka programowania. Prezentuje elementy języka C oraz oparte na klasach, dziedziczeniu i polimorfizmie rozszerzenia obiektowe właściwe dla C++. Takie konstrukcje języka jak pętle, łańcuchy znakowe, tablice, funkcje czy konstruktory klas przedstawiono na możliwie krótkich przykładach. Dla każdego z nich podano kod źródłowy, opis działania, wydruk i zrzut ekranu z wynikami. Szczególną uwagę zwrócono na wyrobienie u Czytelnika sprawności w posługiwaniu się wskaźnikami do łańcuchów, tablic i funkcji. Omówiono przydzielanie i zwalnianie pamięci na stercie przy użyciu operatorów new i delete. Uwzględniono tworzenie przemieszczalnych okienek w trybie tekstowym oraz grafiki punktowej przy wykorzystaniu funkcji oferowanych przez interfejs BGI.

W części drugiej podano kody źródłowe i opisy bardziej zaawansowanych programów. Sześć z nich pracuje w trybie tekstowym a cztery w trybie graficznym. Przykładowo, program Piano symuluje organy a program Mousebox wyświetla sterowane myszą przemieszczalne listy wyboru. W programie City na uwagę zasługuje funkcja takeStr umożliwiająca wpis, przewijanie i modyfikacje łańcucha znakowego dłuższego niż widoczne na ekranie okienko wpisowe. Program Zegar prezentuje dwa przesuwalne strzałkami zegary, z możliwością włączania i wyłączania melodii. Program Drawthru pokazuje użycie myszy do narysowania kilku krzywych wykresu, a następnie do wskazania myszą jednej z nich. W sumie, lektura obydwu części książki winna przybliżyć Czytelnikowi zasady tworzenia programów tak tekstowych jak i graficznych.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 9.

Grafika punktowa

Okno lokalne

Współrzędne punktu na ekranie można opisywać w układzie bezwzględnym dotyczącym całego ekranu lub w podoknie (*viewport*) posiadającym lokalny układ współrzędnych, tak jak ma to miejsce w programie *p9_1.cpp*. W programie tym na początku funkcji `main` uruchamiamy tryb graficzny pracy monitora. Można to uczynić na dwa sposoby: przez dołączenie (wlinkowanie) sterowników grafiki, tak aby sterowniki te stanowiły integralną część naszego programu, lub przez pozostawienie tych sterowników poza naszym programem i korzystanie z nich, w sposób automatyczny, w czasie wykonywania programu. W tym drugim przypadku, używane w programie sterowniki grafiki muszą być obecne na platformie, na której wykonujemy nasz program. Sterowników tych dostarcza środowisko *Borland* w pliku *Bgi*. Uruchomienie trybu graficznego dla tego przypadku realizuje (w naszym programie) funkcja `startG`. Druga możliwość (z „wlinkowaniem", czyli trwałym dołączeniem sterowników grafiki) realizowana jest (w naszym programie) przez funkcję `GWlinkowana`, którą obecnie unieruchamiamy poprzez objęcie jej symbolami `/*...*/`. Ten sposób odwoływania się do sterowników grafiki pokazany zostanie w drugiej części tego podrozdziału.

W funkcji `main` utworzymy dwa prostokątne podokna poprzez dwukrotne wywołanie konstruktora klasy `Okno`. Na liście argumentów tego konstruktora przekazujemy mu parametry `l`, `t`, `r`, `b`, czyli współrzędne wierzchołków: lewego górnego i prawego dolnego kreowanego okna. W obu wywołaniach współrzędne te są różne, tak aby uzyskać wzajemne przesunięcie podokien, jak jest to pokazane na rysunku 9.1. Współrzędne wierzchołków podajemy jako liczby względne odniesione do maksymalnych wymiarów, odpowiednio — w kierunku poziomym i pionowym. Te maksymalne wymiary uzyskujemy z funkcji `getmaxx` oraz `getmaxy`. Klasa `Okno` posiada funkcję własną `VPort`, w której za pomocą funkcji bibliotecznej `setviewport`, ustalamy lokalne układy odniesienia dla obiektów klasy `Okno`: pierwszego, noszącego nazwę `A` i drugiego, wskazywanego przez wskaźnik `pB`. Klasa `Okno` posiada dwie dalsze funkcje własne: `Elips` (do kreślenia elipsy) oraz `Linia` (do wykreślenia prostej). Na rzecz obiektu `A` wywołamy funkcję `Elips`, co spowoduje wykreślenie w pierwszym podoknie (*viewport*) elipsy

usytuowanej centralnie względem tego podokna. Ponieważ jednak przy kreacji tego podokna ostatnim parametrem przekazanym do funkcji `setviewport` była jedynka (z wiersza *36.*), to części figur geometrycznych (w tym przypadku elipsy) wychodzące poza obszar podokna zostaną obcięte. Stąd przycięcie elipsy widoczne na rysunku 9.1. Natomiast na rzecz drugiego okna, wskazywanego przez wskaźnik `pB`, zastosujemy funkcję `Linia`. Ponieważ przy kreacji okna wskazywanego przez wskaźnik `pB` ostatnim parametrem przekazanym funkcji `setviewport` było zero (z wiersza *37.*), więc linia prosta zostanie narysowana w całości (bez ograniczenia jej tylko do obszaru podokna).

Oto program *p9_1.cpp*:

```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>

void startG()
{int ster=DETECT,tryb;
//initgraph(&ster,&tryb,"C:\\BORLANDC\\BGI");
  initgraph(&ster,&tryb,"C:\\Bc5\\Bgi")
 if(graphresult()!=grOk)
 {cprintf("Grafika?"); getch(); exit(1);}
}//___

/*
void GWlinkowana()
{if(registerbgidriver(EGAVGA_driver)<0)
 {cprintf("Grafika: Project ?"); getch(); exit(1);}
 int ster=DETECT,tryb;
 initgraph(&ster,&tryb,"");
 if(graphresult()!=grOk)
 {cprintf("Grafika wlinkowana?"); getch(); exit(1);}
}//___
*/
class Okno
{public: int L,T,R,B,RL,RLp2,BT,BTp2,co,coBk,vP;
 char *O, s[50];
 void VPort();
 void Elips(float rx,float ry);
 void Linia(float x1,float y1,float x2,float y2);
 Okno (float l,float t,float r,float b,char *O,
       int co,int coBk,int vP);
};//_________
void main()
{  startG();
 //GWlinkowana();
 Okno A(.01,.1,.5,.3,"A=",LIGHTGRAY,WHITE,1);
 Okno *pB=&Okno(.4,.2,.7,.4,"B=",YELLOW,LIGHTGRAY,0);
 A.VPort();
 A.Elips(.5, 1.3);
 pB->VPort();
 pB->Linia(.1,.9,.9,-.4);
 getch();  closegraph();
}//_____________________________________________
 Okno::Okno(float l,float t,float r,float b,char *O,
            int co,int coBk,int vP):
```

```
46.  co(co),coBk(coBk),vP(vP),O(O)
47. {int x=getmaxx();
48.  int y=getmaxy();
49.  L=l*x;      T=t*y;
50.  R=r*x;      B=b*y;
51.  RL=R-L;     BT=B-T;
52.  RLp2=RL/2; BTp2=BT/2;
53. }//___
54. void Okno::VPort()
55. {setviewport(L,T,R,B,vP);
56.  setfillstyle(SOLID_FILL,coBk);
57.  bar(0,0,RL,BT);
58.  sprintf(s,"%s%d,%d",O,L,T);
59.  settextjustify(LEFT_TEXT,CENTER_TEXT);
60.  setcolor(BLACK);
61.  outtextxy(2,9,s);
62. }//___
63. void Okno::Elips(float rx,float ry)
64. {setfillstyle(SOLID_FILL,co);
65.  fillellipse(RLp2,BTp2,rx*RLp2,ry*BTp2);
66. }//___
67. void Okno::Linia(float x1,float y1,float x2,float y2)
68. {setlinestyle(SOLID_LINE,0,3);
69.  setcolor(co);
70.  line(x1*RL,y1*BT,x2*RL,y2*BT);
71. }
```

Funkcjonowanie programu obsługującego grafikę punktową wymaga uruchomienia sterownika (*driver*) grafiki, udostępniającego implementacje funkcji pośredniczących pomiędzy programem a konkretnym typem karty graficznej. W tym wypadku chodzi o funkcje dostarczane przez *BGI* (*Borland Graphics Interface*). Funkcje te, na etapie konsolidacji programu, można dołączyć tak, że będą one wraz z programem właściwym stanowić jednolitą całość. Inną możliwość stwarza mechanizm zwany *DLL* (*Dynamic Link Library*) polegający na tym, że program wynikowy nie zawiera modułu z funkcjami z *BGI*. Jednakże moduł ten jest ładowany do pamięci w czasie wykonywania programu, a jego funkcje są udostępniane programowi (a także innym programom ewentualnie wykonywanym równocześnie); stąd oszczędność pamięci i mniejszy rozmiar kodu wynikowego. Inicjalizację grafiki za pomocą drugiego sposobu (*DLL*) realizuje funkcja `startG` w wierszu *6.*, a za pomocą sposobu pierwszego — funkcja `GWlinkowana` w wierszu *15*. Sposób pierwszy jest tu nieaktywny, gdyż funkcja `GWlinkowana` objęta jest znakami `/*...*/`, a jej wywołanie w funkcji `main` jest poprzedzone dwoma ukośnikami (znakami slasz). Uaktywnienia tej funkcji dokonamy w dalszej części rozdziału. Obecnie program graficzny uruchomimy za pomocą `startG` w wierszu *34*.

Inicjalizacji albo otwarcia trybu graficznego dokonuje się poprzez wywołanie funkcji bibliotecznej `initgraph` (wiersz *9.*), przy czym wcześniej należy nadać wartość zmiennej `ster`, tu równą stałej predefiniowanej `DETECT`. Spowoduje to, że funkcja `initgraph` sama rozpozna typ karty graficznej i dobierze dla niej najodpowiedniejszy tryb pracy. Stąd zbędność przypisywania wartości zmiennej `tryb` w wierszu *9*. Jednak, jeśli chcieć uzyskać tzw. stronicowanie ekranu, to zmiennej `ster` należy nadać wartość `VGA`, a zmiennej `tryb` wartość `VGAMED`, jak w programie *ZEGAR.cpp*, w części drugiej. Trzecim argumentem funkcji `initgraph` jest ścieżka dostępu do biblioteki *BGI*.

Może to być np. `C:\BORLANDC\BGI` lub, jak tu, `C:\Bc5\Bgi`. Konieczność używania w programie dwóch wstecznych ukośników `\\` (dwóch znaków *backslash*) wynika stąd, że pierwszy z nich ma (w *C++*) znaczenie specjalne: „potraktuj następny znak dosłownie".

Efektywny jest więc dopiero drugi ukośnik wsteczny `\` (*backslash*). Zawsze należy sprawdzić powodzenie operacji uruchomienia programu graficznego. W tym celu wywołuje się funkcję `graphresult` i sprawdza, czy zwróciła ona wartość predefiniowaną `grOk`.

W wierszu *24.* definiowana jest klasa `Okno` do obsługi jednego wycinka ekranu (podokna lokalnego — *viewport*), z lokalnym układem współrzędnych. Składniki własne `L`, `T`, `R`, `B` oznaczają współrzędne wierzchołków lewego górnego (*Left*, *Top*) i prawego dolnego (*Right*, *Bottom*) podokna lokalnego. Składnik `RL` oznacza wymiar poziomy, a `BT` pionowy okna lokalnego. Składniki własne `RLp2` i `BTp2` są współrzędnymi środka podokna lokalnego (w układzie lokalnym). Składniki `co` i `coBk` zawierać będą informację o kolorze rysunku i tła. Składnik `vP` przyjmować będzie wartość 0 lub 1. W tym ostatnim przypadku części rysunku wystające poza zakres okna lokalnego będą niewidoczne, czyli realizowane będzie *obcinanie* (*clipping*).

W wierszu *26.* widzimy wskaźnik `char *O` łańcucha z częścią tytułu okna. Tablica pomocnicza `s` zawierać będzie cały opis okna (tytuł + współrzędne wierzchołka lewego górnego okna lokalnego).

W wierszu *27.* widzimy funkcję własną `VPort`, zdefiniowaną w wierszu *54*. Funkcja ta, w wierszu *55*, wywołuje funkcję biblioteczną `setviewport` definiującą podokno lokalne. Na liście argumentów aktualnych tej funkcji podajemy współrzędne globalne (dla całego ekranu, z początkiem w lewym górnym rogu ekranu). Po ustaleniu podokna lokalnego, w wierszu *57.* wywołujemy funkcję `bar` (czyli słupek, lub wypełniony prostokąt) podając na liście argumentów współrzędne jego wierzchołków: lewego górnego i prawego dolnego. Współrzędne `0,0` oznaczają ustawienie wierzchołka lewego górnego na początku podokna lokalnego. Współrzędne `RL` i `BT` ustawią prawy dolny wierzchołek słupka (`bar`) w prawym dolnym rogu podokna lokalnego. Słupek (`bar`) wypełniany jest kolorem określanym przez funkcję `setfillstyle` w wierszu *56*.

W wierszu *58.* funkcja `sprintf` dokonuje sformatowanego wpisu do tablicy pomocniczej `s`. Zauważmy, że wpisany do tej tablicy łańcuch będzie zawierał tytuł obiektu wskazany łańcuchem `O` oraz współrzędne `L`, `T` okna lokalnego (względem całego ekranu). Wyprowadzenie tekstu (łańcucha) na ekran, pozostający w trybie graficznym, zapewnia w wierszu *61.* funkcja `outtextxy`. Kolor tego napisu określa funkcja `setcolor` w wierszu *60*. Lokalizację tekstu na ekranie wskazują dwa pierwsze argumenty funkcji `outtextxy`. Centrowanie tekstu wokół tej lokalizacji zapewnia funkcja `settextjustify` w wierszu *59*.

W wierszu *28.* widzimy funkcję własną `Elips` zdefiniowaną w wierszu *63*. Funkcja ta, w wierszu *65*, wywołuje funkcję biblioteczną `fillellipse` dla argumentów będących składnikami własnymi klasy, z których `RLp2` oznacza połowę szerokości, a `BTp2` połowę wysokości okna lokalnego. Pozostałe dwa argumenty są długościami półosi elipsy (przeliczanymi z jednostek względnych `rx` i `ry` na bezwzględne, w pikselach).

W wierszu *29.* widzimy następną funkcję własną o nazwie `Linia`, zdefiniowaną w wierszu *67*. Funkcja ta wywołuje funkcję biblioteczną `line` z lokalnymi współrzędnymi początku i końca linii prostej. Instrukcje przeliczające jednostki względne `x1`, `x2`, `y1`, `y2` na lokalne bezwzględne (w pikselach) występują na liście argumentów aktualnych funkcji `line` w wierszu *70*. Funkcja `setlinestyle` w wierszu *68.* określa rodzaj linii (ciągła, przerywana, itp.) oraz jej grubość. Wartość 3 ostatniego argumentu oznacza żądanie kreślenia linii pogrubionej.

W wierszu *30.* widzimy konstruktor klasy `Okno`, zdefiniowany w wierszu *44*. Na liście parametrów formalnych otrzymuje on informację `l`, `t`, `r`, `b` o położeniu wierzchołków (lewego górnego i prawego dolnego) w jednostkach względnych globalnych, takich że zero odpowiada lewemu górnemu, a jeden — prawemu dolnemu rogowi ekranu. W ciele konstruktora współrzędne te są przeliczane na współrzędne bezwzględne globalne `L`, `T`, `R`, `B`. W tym celu w wierszu *47.* (i następnym) wywołujemy funkcje `getmaxx` i `getmaxy` zwracające wartości maksymalne (liczone w pikselach) współrzędnych ekranu w poziomie i w pionie. Kolory, `vP` (*clipping*) oraz łańcuch `O` tytułu obiektu przekazywane są do klasy za pośrednictwem listy inicjalizacyjnej (przed wierszem *47.*).

W programie `main` wywołujemy funkcję inicjalizującą grafikę albo z wiersza *34.* (obecnie), albo z wiersza *35.* (w drugiej części niniejszego podrozdziału).

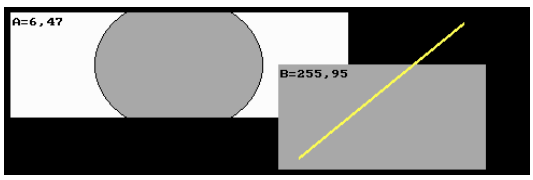
W wierszu *36.* kreowany jest obiekt o nazwie `A` poprzez wywołanie na jego rzecz konstruktora klasy `Okno`. W wierszu *37.* powtarzamy czynności z wiersza *36.*, ale dla obiektu `B` klasy `Okno`, z tym jednak że tu (dla odmiany) wykreowany obiekt wskazywany jest nie nazwą, ale wskaźnikiem `pB`. Zauważmy, że ostatni argument w wywołaniu konstruktora wynosi: 1 w wierszu *36.* i 0 w wierszu *37*. Zatem, w przypadku obiektu `A` będzie działać mechanizm, który spowoduje obcinanie (*clipping*) wychodzących poza podokno lokalnych części rysowanych figur geometrycznych lub napisów. W przypadku okna `B` tak nie będzie.

W wierszu *38.* wywołujemy funkcję `Vport`, a w wierszu *39.* funkcję `Elips` na rzecz obiektu `A`. Ponieważ `A` jest nazwą obiektu, więc w wierszach *38.* i *39.* używamy operatora kropki.

W wierszach *40.* i *41.* wywołujemy funkcje `VPort` oraz `Linia` na rzecz obiektu `B` wskazywanego wskaźnikiem `pB`. Stąd konieczność użycia operatora strzałki w tych wywołaniach funkcji.

Przed zakończeniem programu należy zamknąć tryb graficzny poprzez wywołanie funkcji `closegraph` w wierszu *42*.

Po skompilowaniu, konsolidacji i wykonaniu programu w trybie *Windows* lub *MS-DOS* otrzymujemy ekran, jaki przedstawiony został na rysunku 9.1.



Rysunek 9.1. *Ekran otrzymany w wyniku wykonania programu p9_1.exe*

Podokno pierwsze pracuje w trybie obcinania figur wystających poza obszar podokna lokalnego, a okno drugie w trybie bez obcinania.

Zauważmy, że elipsa obiektu `A` została obcięta na granicy podokna lokalnego ustalanego funkcją biblioteczną `setviewport` w wierszu *55.* programu. Ponieważ mechanizmu tego nie wywołano przy kreacji obiektu `B` (w wierszu *37.*), więc linia narysowana na rzecz tego obiektu wychodzi poza granice okna lokalnego (*viewport*).

Wykonanie powyższego programu na innym komputerze jest możliwe tylko wtedy, gdy sterowniki grafiki *EGAVGA* znajdują się również w katalogu *C:\Bc5\Bgi*, tzn. takim jak w wierszu *9.* programu. W przeciwnym razie należy ponownie skompilować program, uaktualniając ścieżkę dostępu w wierszu *9.* programu. Innym rozwiązaniem, nie wymagającym rekompilacji, byłoby podanie w wierszu *9.* programu pustej ścieżki dostępu do sterowników grafiki (czyli `""`). Wtedy program (w czasie wykonania) poszukuje sterowników w katalogu bieżącym. Mając tak skompilowany program, przed jego wykonaniem należałoby skopiować plik *Egavga* (z zasobów *Borland*) do bieżącego katalogu (w którym znajduje się *p9_1.exe*) i wtedy dopiero go wykonać.

Drugi sposób udostępniania programowi sterowników grafiki (tzn. sposób z tzw. wlinkowaniem czyli trwałym dołączeniem sterowników *EGAVGA*) pokażemy w programie *p9_1wg.cpp* (odmiana programu *p9_1.cpp*) zlokalizowanym w katalogu *p9_1wg* (o takiej samej nazwie jak program).

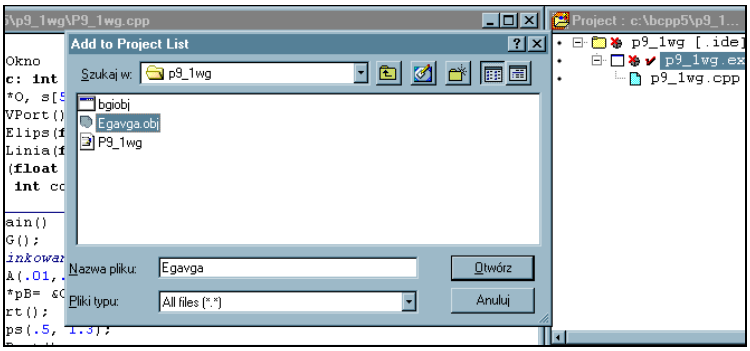
Najpierw należy uzyskać plik *Egavga.obj*. W tym celu do bieżącego katalogu, np. *C:\Bcpp5\p9_1wg*, należy skopiować program *Bgiobj.exe* (z zasobów *Borland*), np. z katalogu *C:\Bc5\Bgi*. Kopiowanie to przeprowadzamy poza środowiskiem programowania *Borland*, np. wykorzystując program Eksplorator Windows.

Następnie (nadal korzystając z programu Eksplorator Windows) otwieramy bieżący katalog, po czym wskazujemy myszą podane dalej pola i wpisujemy: *Start|Uruchom|Bgiobj C:\Bg5\Bgi\Egavga*. Po wykonaniu i zamknięciu programu *Bgiobj.exe*, w katalogu *C:\Bcpp5\p9_1wg* powinien się pojawić plik *Egavga.obj*. Zamykamy program Eksplorator Windows.

Uruchamiamy środowisko programowania *Borland C++* i tworzymy nowy projekt o nazwie *p9_1wg*, dla platformy *DOS* w katalogu *p9_1wg*, z tym jednak że w oknie *New Target* zaznaczamy kwadratowe pole *BGI* (rysunek 2.2).

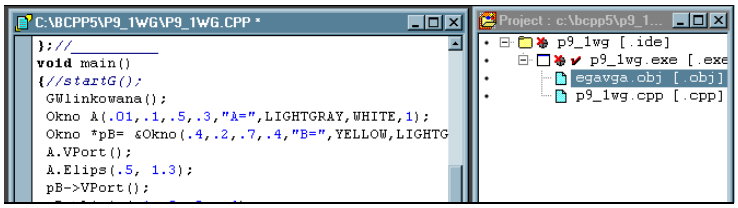
Wskazując myszą na ikony *File|Open* odszukujemy program *p9_1.cpp* (w katalogu *p9_1*) i zapisujemy go w katalogu *p9_1wg* pod nazwą *p9_1wg.cpp*.

W oknie *Project* naprowadzamy kursor na kwadrat z minusem tuż przed nazwą *p9_1wg.exe* i naciskamy prawy przycisk myszy. Pojawia się lista wyboru, na której lewym przyciskiem myszy wskazujemy polecenie *Add node*. Pojawi się okno *Add to Project List*. W polu Pliki typu rozwijamy listę i wybieramy opcję *All files* (*.*). Myszą wskazujemy na *Egavga.obj*, jak pokazano na rysunku 9.1a.



Rysunek 9.1a. *Okno środowiska programowania Borland w czasie dodawania do już utworzonego projektu (w bieżącym katalogu) pliku Egavga.obj w celu trwałego dołączenia sterowników grafiki*

Po wskazaniu myszą przycisku *Otwórz*, okno *Add to Project List* znika, a w oknie *Project* pojawia się nowy węzeł (*node*) *egavga.obj*, jak pokazano na rysunku 9.1b.



Rysunek 9.1b. *Fragment okna środowiska programowania Borland po dodaniu do projektu pliku Egavga.obj*

Uaktywniamy (poprzez kliknięcie) okno z plikiem programu *p9_1wg.cpp* i dokonujemy w nim modyfikacji polegającej na uruchomieniu funkcji `GWlinkowana` zamiast `startG`. Po skompilowaniu i konsolidacji otrzymujemy program, którego integralną częścią są sterowniki grafiki *EGAVGA*. Wykonanie tego programu prowadzi do takiego samego wydruku, jak wydruk przedstawiony na rysunku 9.1.

Pozycjonowanie obiektu za pomocą myszy

Do przesuwania obiektów takich jak figury geometryczne można użyć myszy, podobnie jak zostanie to zrobione w programie *p9_2.cpp*. W programie tym dostęp do sterowników grafiki realizujemy sposobem przedstawionym w drugiej części poprzedniego podrozdziału, tzn. poprzez wywołanie funkcji `GWlinkowana`. (Funkcja `startG` jest tu nieczynna). W tym celu uruchamiamy projekt w katalogu *p9_2* z zaznaczeniem pola BGI (widocznego na rysunku 2.2) i z dodaniem węzła (*node*) w postaci pliku *egavga.obj* (jak na rysunku 9.1b). Plik ten można np. przegrać z katalogu *p9_1wg* lub wskazać go w tym katalogu w czasie dołączania nowego węzła.

Oczywiście można go też wykreować od nowa za pomocą programu *Bgiobj.exe*, tak jak zrobiono to w drugiej części poprzedniego podrozdziału.

Do obsługi myszy zdefiniujemy klasę `Mysz` zawierającą funkcje własne: `iniM` (do uruchomienia myszy), `pokM` (do ujawniania kursora myszy), `ukrM` (do ukrycia kursora myszy) i `gdzM` (do zwrotu aktualnych współrzędnych kursora myszy). W funkcji `main` utworzymy tylko jeden obiekt o nazwie `A` klasy (typu) `Okno`. Obiekt ten jest widocznym na rysunku 9.2a prostokątem o wymiarach zdefiniowanych za pomocą wartości argumentów w wywołaniu konstruktora klasy `Okno` w wierszu *49*. Na rzecz tego obiektu wywołujemy funkcję własną `VPort` klasy `Okno`. W funkcji tej definiujemy podokno z lokalnym układem współrzędnych, pokrywające się z utworzonym właśnie prostokątem `A`. W tym podoknie narysujemy elipsę oraz w prawym dolnym rogu mały kwadrat, pełniący rolę uchwytu do przesuwania okna. Następnie w funkcji `main` uruchomimy nieskończoną pętlę `while(1)`, w której sprawdzimy, czy kursor myszy znajduje się w polu uchwytu przesuwu i czy wtedy naciśnięty jest lewy przycisk myszy. Jeśli tak, to dotychczasowe podokno (*viewport*) ulega wymazaniu, a cały rysunek przerysowywany jest w nowym (wskazanym myszą) położeniu. W tym nowym położeniu ponownie ustalimy lokalny układ współrzędnych (nowy *viewport*). Naciśnięcie klawisza *Esc* przerywa nieskończoną pętlę `while` i program kończy działanie.

Oto program *p9_2.cpp*:

```
 1. #include <stdlib.h>
 2. #include <conio.h>
 3. #include <graphics.h>
 4. #include <stdio.h>
 5. #include <dos.h>
 6. /*
 7. void startG()
 8. {int ster=DETECT,tryb;
 9.  initgraph(&ster,&tryb,"C:\\BORLANDC\\BGI");
10.  if(graphresult()!=grOk)
11.  {cprintf("Grafika?"); getch(); exit(1);}
12. }//___
13. */
14.
15. void GWlinkowana()
16. {if(registerbgidriver(EGAVGA_driver)<0)
17.  {cprintf("Grafika: Project ?"); getch(); exit(1);}
18.  int ster=DETECT,tryb;
19.  initgraph(&ster,&tryb,"");
20.  if(graphresult()!=grOk)
21.  {cprintf("Grafika wlinkowana?"); getch(); exit(1);}
22. }//___
23.
24. class Okno
25. {public:
26.  int L,T,R,B,RL,RLp2,BT,BTp2,co,coVP,coVPBis,vP,
27.      xMax,yMax;
28.  void VPort(int coBis=0);
29.  void NewVP();
30.  void Elips(float rx,float ry);
31.  Okno (float l,float t,float r,float b,char *O,
32.        int co,int coBk,int coBkBis,int vP);
33. };//_________
34.
35. class Mysz
```

```
{public: char S[20]; union REGS R;
 int x,y,xO,yO,Bu,BuO,Nxy,NBu;
 void iniM();
 void pokM() {R.x.ax=1; int86(0x33,&R,&R);}
 void ukrM() {R.x.ax=2; int86(0x33,&R,&R);}
 void gdzM();
};//_____________________________________
Mysz M;

void main()
{int k,ll,tt,rr,bb,nvp; //startG();
    GWlinkowana();
 M.iniM(); M.pokM();
 Okno A(.1,.1,.5,.3,RED,LIGHTGRAY,WHITE,1);
 A.VPort();
 while(1)
 {k=0; if(kbhit()) if(!(k=getch())) k=getch();
  if(k==27) break;
  M.gdzM();
  if(M.x>A.R-10 && M.x<A.R && M.y>A.B-10 && M.y<A.B)
  {M.xO=M.x; M.yO=M.y; M.BuO=M.Bu; nvp=1;
   while(M.Bu==1)
   {if(nvp){A.NewVP(); nvp=0;}
    M.gdzM();
    if(M.Nxy)
    {ll=M.x-(A.RL-5);tt=M.y-(A.BT-5);rr=M.x+5;bb=M.y+5;
     if(ll<0) ll=0;
     if(tt<0) tt=0;
     if(rr>A.xMax) ll=A.xMax-A.RL;
     if(bb>A.yMax) tt=A.yMax-A.BT;
     A.L=ll; A.T=tt; A.R=A.L+A.RL; A.B=A.T+A.BT;
     A.NewVP();
   }}
   if(M.Nxy || M.NBu) A.VPort();
 }}
 getch(); M.ukrM(); closegraph();
}//_____________________________________

Okno::Okno(float l,float t,float r,float b,
       int co,int coVP,int coVPBis,int vP):
 co(co),coVP(coVP),coVPBis(coVPBis),vP(vP),
{xMax=getmaxx(); yMax=getmaxy();
 L=l*xMax; T=t*yMax; R=r*xMax;  B=b*yMax;
 RL=R-L;   BT=B-T;   RLp2=RL/2; BTp2=BT/2;
}//___

void Okno::VPort(int coBis)
{setviewport(L,T,R,B,vP);
 int col= (coBis)? coVPBis:coVP;
 setfillstyle(SOLID_FILL,col);
 M.ukrM(); bar(0,0,RL,BT);  Elips(.8,.5);
 setfillstyle(SOLID_FILL,YELLOW);
 bar(RL-10,BT-10,RL,BT);
   sprintf(M.S,"%3d,%3d",M.x,M.y);
   setfillstyle(SOLID_FILL,RED);
   bar(4,3,65,15); outtextxy(6,6,M.S);
 M.pokM();
```

```
 93. }//___
 94.
 95. void Okno::NewVP()
 96. {M.ukrM(); clearviewport(); M.pokM(); VPort(1);
 97. }
 98. void Okno::Elips(float rx,float ry)
 99. {setfillstyle(SOLID_FILL,co);
100.  fillellipse(RLp2,BTp2,rx*RLp2,ry*BTp2);
101. }//___
102. void Mysz::iniM()
103. {R.x.ax=0; int86(0x33,&R,&R); if(R.x.ax==0)
104.  {cprintf("\n Zainstaluj mysz"); getch(); exit(0);}
105.  gdzM(); xO=x; yO=y;
106. }
107.
108. void Mysz::gdzM()
109. {R.x.ax=3; int86(0x33,&R,&R); x=R.x.cx; y=R.x.dx;
110.  Bu=(R.x.bx & 0x0001)? 1:0;
111.  Nxy= (x!=xO || y!=yO)? 1:0;  NBu= (Bu!=BuO)? 1:0;
112.  if(Nxy) {xO=x; yO=y;}
113. }
```

W wierszu *7.* widzimy początek funkcji `startG` uruchamiającej grafikę bez dołączania sterowników z *BGI*. Funkcja ta jest tu nieczynna z powodu objęcia jej ciała znakami `/*  */`. W wierszu *15.* widzimy początek definicji funkcji `GWlinkowana` inicjalizującej tryb graficzny z trwałym dołączeniem sterownika *EGAVGA* z *BGI*. Ta funkcja jest tu aktywna. Przed kompilacją programu należy utworzyć plik *Bgiobj.obj* (jak w programie *p9_1gw.cpp*) i dołączyć go do listy węzłów (*node*) w oknie *Project*.

W wierszu *24.* widzimy początek definicji klasy `Okno`. Składnikami własnymi tej klasy są współrzędne przeciwległych wierzchołków okna lokalnego oraz jego wymiary. Zmienne `xMax` i `yMax` przechowywać będą informację o maksymalnych wartościach współrzędnych ekranu w poziomie i w pionie w aktualnym trybie karty graficznej.

W wierszu *28.* funkcja `VPort`, ustalająca okno lokalne, posiada jeden argument z wartością domniemaną `coBis = 0`. Wartość 1 tego argumentu spowoduje wybranie dla okna lokalnego alternatywnego koloru (wtedy, gdy okno to podlega przesuwaniu w inne miejsce). Realizuje to funkcja `NewVP` w wierszu *95.*, wywołująca funkcję `VPort` z parametrem `coBis = 1`. Jednakże, gdy nie ma przesuwania okna, wybierany jest zwykły kolor określony zmienną `co`, co odpowiada podanej wartości domniemanej zmiennej `coBis` równej zero.

Deklarację funkcji własnej `NewVP` widzimy w wierszu *29.*, a jej definicję w wierszu *95*. Przed narysowaniem nowego okna wywoływana jest funkcja `clearviewport` czyszcząca okno dotychczasowe. Funkcja `cleardevice` wyczyściłaby cały ekran.

W wierszu *32.* widzimy deklarację konstruktora klasy `Okno`. W wierszu *35.* widzimy początek klasy `Mysz` zapewniającej obsługę myszy. Składnik własny w postaci tablicy znakowej `S` służyć będzie do formatowania wydruku (za pomocą funkcji `sprintf`) z aktualnymi współrzędnymi myszy. Składniki `x`, `y` przechowują współrzędne aktualne, a `xO` i `yO` współrzędne poprzednie kursora myszy. Składniki `Bu` i `BuO` podają informację o aktualnym i poprzednim naciśnięciu lewego przycisku myszy. Jeśli nastąpiła zmiana

współrzędnych lub zmiana stanu naciśnięcia przycisku myszy, to składniki `Nxy` lub `NBu` wynoszą jeden. Funkcja `iniM` zadeklarowana w wierszu *38.* jest zdefiniowana w wierszu *102.* i następnych. W ciele tej funkcji widzimy wywołanie funkcji 0H przerwania 33H oraz sprawdzenie poprawności inicjalizacji. Ewentualne niepowodzenie powoduje wydruk łańcucha `Zainicjalizuj mysz`. Należy wtedy odszukać w zasobach komputera np. funkcję *msmouse.com* i ją wykonać. Można wtedy ponownie przystąpić do wykonania programu *p9_2.exe*.

W wierszu *39.* podano deklarację i definicję funkcji `pokM` ujawniającej kursor myszy. Funkcja ta wywołuje funkcję 1H przerwania 33H. Ukrywanie kursora myszy zapewnia funkcja `ukrM` z wiersza *40*. Wywołuje ona funkcję 2H przerwania 33H.

W wierszu *41.* widzimy deklarację funkcji `gdzM` dostarczającej informacji o aktualnym położeniu myszy. Jej definicja znajduje się w wierszu *108.* i następnych. Wywołuje ona funkcję 3H przerwania 33H. Przerwanie to zwraca współrzędne myszy w rejestrach *CX* i *DX*. Informacja o stanie przycisku myszy w chwili wywoływania przerwania znajduje się w rejestrze *BX*. Informacje te przekazywane są do składników własnych klasy `Mysz`. W wierszu *111.* ustalane są składniki niosące informację o tym, czy współrzędne myszy się zmieniły, lub czy zmienił się stan naciśnięcia przycisku myszy.

W wierszu *43.* (jeszcze przed `main`) definiowany jest obiekt `M` klasy `Mysz`. Obiekt ten ma zasięg globalny, gdyż jest zdefiniowany poza ciałem jakiejkolwiek funkcji.

W wierszu *46.* widzimy wywołanie funkcji `startG`, z tym że jest ono nieaktywne (po dwóch ukośnikach —znakach slasz). Aktywne jest wywołanie funkcji `GWlinkowana` w wierszu *47*.

W wierszu *48.* inicjalizujemy i ujawniamy mysz. W wierszu *49.* kreowany jest obiekt `A` klasy `Okno` poprzez wywołanie konstruktora tej klasy. Na liście parametrów aktualnych podajemy współrzędne względne globalne (odnoszące się do całego ekranu) wierzchołków lewego górnego i prawego dolnego okna lokalnego. Na rzecz tego obiektu w wierszu *50.* wywoływana jest funkcja `VPort` ustalająca lokalny układ współrzędnych.

W wierszu *51.* zaczyna się zasadnicza część programu w postaci nieskończonej pętli `while(1)`. W wierszu *52.* sprawdzamy, czy bufor klawiatury jest niepusty. Jeśli tak, i jeśli naciśniętym klawiszem był klawisz *Esc* (o kodzie *ASCII* = 27), to pętla ta ulega przerwaniu, po czym następuje zakończenie programu. W wierszu *54.* sprawdzamy aktualne współrzędne myszy wywołując funkcję `gdzM` na rzecz obiektu `M` o zasięgu globalnym.

W wierszu *55.* sprawdzamy, czy kursor myszy znajduje się na prawym dolnym uchwycie do przesuwania okna. Jeśli tak, to wykonywane jest całe ciało instrukcji `if` od wiersza *55.* do *68*. W wierszu *57.* sprawdzamy dodatkowo, czy lewy przycisk myszy jest naciśnięty. Jeśli tak, to wykonywane jest całe ciało instrukcji `while(M.Bu==1)`, od wiersza *57.* do *68*. W wierszu *58.* wywoływana jest funkcja `NewVP` po to, aby zmienić kolor okna przy przesuwaniu. W wierszu *59.* odczytujemy współrzędne bieżące myszy (w trakcie przesuwania). W wierszu *60.* sprawdzamy, czy współrzędne te są inne

niż poprzednio. Jeśli tak, to wykonywane jest ciało instrukcji warunkowej `if(M.Nxy)` w wierszach *62.* i *67.* W wierszu *67.* za pomocą funkcji `NewVP` rysowane jest okno lokalne, z tym że wiersze poniżej wiersza *62.* zabezpieczają brzegi okna przed wyjściem poza zakres ekranu.

Jeśli przycisk myszy zostanie zwolniony, to przerywana jest pętla `while(M.Bu==1)` i w wierszu *69.* funkcja `VPort`, wywołana na rzecz obiektu `A`, odtwarza normalny kolor okna lokalnego.

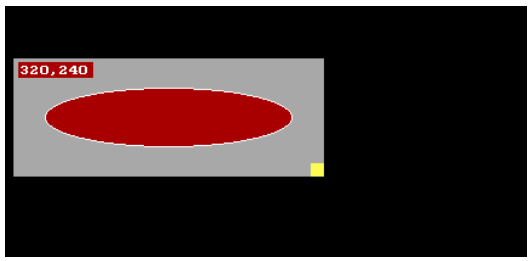
Po przerwaniu pętli `while(1)` za pomocą instrukcji `break` z wiersza *52.* program przechodzi do wiersza *71.* i, po zakończeniu trybu graficznego poprzez wywołanie funkcji `closegraph`, `ulega zamknięciu`.

W wierszu *74.* widzimy definicję konstruktora klasy `Okno`. W wierszu *77.* wymiary ekranu (w pikselach) wpisywane są do zmiennych `xMax` i `yMax`.

Funkcja własna `VPort` w wierszu *82.* oczekuje na liście parametrów formalnych informacji o tym, czy do rysowania okna lokalnego należy używać alternatywnego koloru. Wartość domniemana tego parametru wynosi zero (wiersz *28.*). Na jego podstawie w wierszu *84.* ustalany jest kolor w zmiennej lokalnej `col`. W wierszu *86.* rysujemy „prostokąt" (za pomocą funkcji bibliotecznej `bar`) oraz elipsę, wywołując funkcję `Elips` z wiersza *98.* W wierszu *88.* rysowany jest uchwyt do przesuwania okna. W wierszu *89.* funkcja `sprintf` wpisuje współrzędne myszy do łańcucha pomocniczego `S` w obiekcie `M` klasy `Mysz`. Łańcuch ten jest w wierszu *91.* wyświetlany na ekranie za pomocą funkcji `outtextxy`.

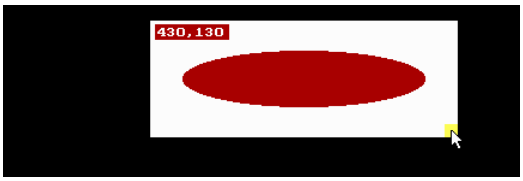
Każda operacja graficzna na ekranie powinna być wykonywana przy ukrytej na ten czas myszy. Po wykonaniu tych operacji ujawniamy mysz, jak w wierszu *92.*, wywołując funkcję `pokM` na rzecz obiektu `M`.

Po skompilowaniu, konsolidacji i uruchomieniu programu w trybie *Windows* lub *MS-DOS* otrzymujemy ekran widoczny na rysunku 9.2a.



Rysunek 9.2a. *Ekran programu p9_2.exe z prostokątem, elipsą, informacją o położeniu kursora i uchwytem do przesuwu całego rysunku za pomocą myszy z naciśniętym lewym przyciskiem*

W lewym górnym rogu okna widzimy aktualne współrzędne myszy. W prawym dolnym rogu widzimy uchwyt do przesuwania prostokątnego okna lokalnego z elipsą w jego centrum. W trakcie przesuwania okna (ciągnąc mysz zaczepioną w uchwycie) zmianie ulega kolor tła okna lokalnego, jak przedstawiono to na rysunku 9.2b.



Rysunek 9.2b. *Ekran programu p9_2.exe w czasie przesuwania za pomocą myszy lokalnego okna w nowe położenie*

Zwolnienie przycisku myszy przywraca oryginalne kolory okna lokalnego. W celu zakończenia programu należy nacisnąć klawisz *Esc*.

Kopiowanie wycinków obrazu do pamięci

W wyniku wykonania programu *p5_6.exe* utworzony został zbiór *mojPlik* z kodami *ASCII* dziesięciu znaków. Obecnie wartości tych kodów potraktujemy jako rzędne dziesięciopunktowego wykresu z rysunku 9.3a. Plik *mojPlik* kopiujemy z katalogu *p5_6* do katalogu bieżącego *p9_3*, w którym utworzymy obecny projekt dla programu *p9_3.cpp* — z grafiką wlinkowaną (trwale dołączoną) za pomocą znanej z poprzedniego programu funkcji `GWlinkowana`. W wierszu *43.* utworzymy obiekt `A` poprzez wywołanie konstruktora klasy `Okno`, który jest prostokątem koloru jasnoszarego (`LIGHTGRAY`). Na rzecz obiektu `A` wywołamy funkcję `VPort`, ustalającą aktualny układ odniesienia pokrywający się z prostokątem `A`. Mając już układ odniesienia, wywołujemy na rzecz obiektu `A` funkcję `Kresl`. W funkcji tej stworzymy dostęp do pliku *mojPlik* (używając funkcji bibliotecznej `fopen`) i odczytamy z tego pliku dane do wykresu. Wartości kodów *ASCII* kolejno odczytanych znaków stanowią rzędne dla wierzchołków linii łamanej tworzonego wykresu. Wykres utworzymy poprzez wywołanie funkcji bibliotecznej `line` dla każdego wierzchołka linii łamanej. W wierzchołkach tych narysujemy małe kółka (używając funkcji bibliotecznej `pieslice`). Następnie, w `main` uruchomimy nieskończoną pętlę `while(1)`. W pętli tej sprawdzamy położenie kursora myszy i ewentualne naciśnięcie jej lewego przycisku. Jeśli kursor naprowadzimy na jeden z wierzchołków linii łamanej (na jedno z kółek) i naciśniemy przycisk myszy, to nad wskazanym wierzchołkiem pojawi się prostokąt z napisem (rysunku 9.3b) zawierającym informację o znaku i kodzie *ASCII*, jakiemu odpowiada dany wierzchołek wykresu (czyli dane kółko).

Po zwolnieniu przycisku myszy, napis powinien zostać usunięty, a wykres powinien przyjąć oryginalną postać. Aby nie naruszać pozostałej części wykresu, postulat ten zrealizujemy w ten sposób, że jeszcze przed wyświetleniem napisu umieścimy w pamięci (używając funkcji bibliotecznej `getimage`) oryginalny obraz prostokątnego wycinka ekranu, w którym za moment ma pojawić się napis. Odtworzenie wykresu (po zakończeniu wyświetlania napisu) polegać będzie na skopiowaniu w miejsce napisu obrazu wycinka ekranu, umieszczonego uprzednio w pamięci. Zostanie to zrealizowane za pomocą funkcji bibliotecznej `putimage`.

Oto program *p9_3.cpp*:

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

void GWlinkowana()
{if(registerbgidriver(EGAVGA_driver)<0)
 {cprintf("Grafika: Project ?"); getch(); exit(1);}
 int ster=DETECT,tryb;
 initgraph(&ster,&tryb,"");
 if(graphresult()!=grOk)
 {cprintf("Grafika wlinkowana?"); getch(); exit(1);}
}//___

class Okno
{public:
 int L,T,R,B,RL,BT,co,coVP,
 xMax,yMax, fY[10], XX[10],YY[10],
 Xl[10],Xp[10],Yg[10],Yd[10],w,h,wym;
 static double puX[],puY[]; char s[10]; void *pIm;
 void VPort(); void Kresl();
 Okno (float l,float t,float r,float b,
       int co,int coVP);
};//_________

class Mysz
{public: char S[20]; union REGS R;
 int x,y,xO,yO,Bu;
 void iniM();
 void pokM() {R.x.ax=1; int86(0x33,&R,&R);}
 void ukrM() {R.x.ax=2; int86(0x33,&R,&R);}
 void gdzM();
};//____
Mysz M;
double Okno::puX[]={0.,.03,0.,0.,1.,.9,.9},
       Okno::puY[]={.8,.8,1.,0.,0.,.05,0.};//_____

void main()
{int i,k, wPam, u,v; //startG();
                      GWlinkowana();
 M.iniM(); M.pokM();
 Okno A(.1,.1,.7,.4,RED,LIGHTGRAY);
 A.VPort();
 A.Kresl();
 while(1)
 {k=0; if(kbhit()) if(!(k=getch())) k=getch();
  if(k==27) break;
  M.gdzM();
  for(i=0;i<10;i++)
  {if(A.Xl[i]<M.x && M.x<A.Xp[i] &&
      A.Yg[i]<M.y && M.y<A.Yd[i] && M.Bu)
   {wPam=0;
    do{if(!wPam)
       {setfillstyle(SOLID_FILL,YELLOW);
```

```
56.         setcolor(BLUE);
57.         sprintf(A.s," %c,%3d ",A.fY[i],A.fY[i]);
58.         u=A.XX[i]-A.w/2;
59.         v=A.Yg[i]-A.h-A.T;
60.         M.ukrM();
61.         getimage(u,v, u+A.w,v+A.h,A.pIm);
62.         bar(u,v,u+A.w,v+A.h); outtextxy(u,v,A.s);
63.         M.pokM(); wPam=1;
64.        }
65.        M.gdzM();
66.       }while(M.Bu);
67.    }
68.    if(wPam)
69.    {M.ukrM(); putimage(u,v,A.pIm,COPY_PUT);
70.     M.pokM(); wPam=0;
71.  }}}
72.  M.ukrM(); closegraph(); delete[] A.pIm; A.pIm=NULL;
73. }//_____________________________________________
74.
75. Okno::Okno(float l,float t,float r,float b,
76.       int co,int coVP): co(co),coVP(coVP)
77. {xMax=getmaxx(); yMax=getmaxy();
78.  L=l*xMax; T=t*yMax; R=r*xMax;  B=b*yMax;
79.  RL=R-L;   BT=B-T;
80.  for(int i=0;i<7;i++)
81.  {puX[i]=puX[i]*RL;  puY[i]=(1.-puY[i])*BT;}
82.  sprintf(s," %c,%3d ",'a',999);
83.  w=textwidth(s); h=textheight(s);
84.  wym=imagesize(0,0,w,h); pIm=new char[wym];
85.  if(!pIm){M.ukrM(); closegraph();
86.  cprintf(" new pIm?"); getch(); exit(0);}
87. }//___
88.
89. void Okno::VPort()
90. {setviewport(L,T,R,B,1);
91.  setfillstyle(SOLID_FILL,coVP);
92.  M.ukrM(); bar(0,0,RL,BT);
93.  setlinestyle(SOLID_LINE,0,3); setcolor(WHITE);
94.  for(int i=1;i<7;i++)
95.  line((int)puX[i-1], (int)puY[i-1],
96.       (int)puX[i],   (int)puY[i]);
97.  setcolor(RED); outtextxy(270,10,"Kliknij punkt");
98.  M.pokM();
99. }//___
100.
101. void Okno::Kresl()
102. {int i=1,j, x,y,dX; FILE *p=fopen("mojPlik","rt");
103.  if(p==NULL)
104.  {outtextxy(10,10,"mojPlik ?"); getch(); exit(0);}
105.  rewind(p);
106.  setcolor(co); setfillstyle(SOLID_FILL,co);
107.  fY[0]=fgetc(p);
108.  dX=RL/11; x=dX; y=(110.-fY[0])/110.*BT;
109.  XX[0]=x; YY[0]=y;
110.  do{fY[i]=fgetc(p);
111.     j=(110.-fY[i])/110.*BT;
112.     line(x,y, x+dX,j); pieslice(x,y,0,360,4);
113.     x+=dX; y=j; XX[i]=x; YY[i]=y;
```

```
114.    }while(j!=EOF && ++i<10);
115.    pieslice(x,y,0,360,4);
116.    for(i=0;i<10;i++)
117.    {Xl[i]=L+XX[i]-4; Xp[i]=L+XX[i]+4;
118.     Yg[i]=T+YY[i]-4; Yd[i]=T+YY[i]+4;
119.    }
120. }//___
121.
122. void Mysz::iniM()
123. {R.x.ax=0; int86(0x33,&R,&R); if(R.x.ax==0)
124.  {cprintf("\n Zainstaluj mysz"); getch(); exit(0);}
125.  gdzM(); xO=x; yO=y;
126. }
127.
128. void Mysz::gdzM()
129. {R.x.ax=3; int86(0x33,&R,&R); x=R.x.cx; y=R.x.dx;
130.  Bu=(R.x.bx & 0x0001)? 1:0;
131.  if(x!=xO || y!=yO) {xO=x; yO=y;}
132. }
```

W wierszu *7.* rozpoczyna się definicja funkcji uruchamiającej tryb graficzny (jak w programie *p9_2.cpp*). W wierszu *16.* rozpoczyna się definicja klasy `Okno`. Składniki własne tej klasy (w wierszu *18.*) zawierać będą położenia i wymiary okna lokalnego. W wierszu *19.* widzimy dwie tablice na pomieszczenie współrzędnych dziesięciopunktowego (dziesięciowęzłowego) wykresu. Będą to współrzędne bezwzględne (w pikselach) w oknie lokalnym. Każdy węzeł wykresu jest kołem o promieniu równym cztery piksele. We wnętrzu każdego z tych kół wyróżnimy kwadrat, taki że jeśli współrzędne aktualne kursora myszy znajdą się we wnętrzu tego kwadratu, to dany węzeł uznamy za wskazany (zidentyfikowany) myszą. Lewe i prawe oraz górne i dolne brzegi takich kwadratów (dla każdego węzła) zapisane zostaną w tablicach `Xl`, `Xp`, `Xg` i `Xd`, w wierszu *20*. W wierszu *21.* widzimy tablice `puX` i `puY` o nie zdefiniowanych rozmiarach. W tablicach tych zapiszemy (początkowo w jednostkach względnych lokalnych) współrzędne tak dobranych punktów, że po połączeniu ich linią łamaną zostaną wyrysowane osie lokalnego układu odniesienia (razem z półstrzałkami). Ponieważ deklaracja tych tablic poprzedzona jest deskryptorem `static`, więc tablice te należy zainicjalizować globalnie, poza ciałem jakiejkolwiek funkcji. Umożliwi to ustalenie rozmiarów tych tablic na etapie inicjalizacji (w wierszu *36.* i następnym) na podstawie liczby wypisanych po prawej stronie elementów danej tablicy.

W wierszu *21.* widzimy również deklarację wskaźnika `pIm` obiektu typu `void`, tzn. obiektu typu nie zdefiniowanego. Wskaźnik ten będzie wskazywał na uprzednio zarezerwowany (wiersz *83.*) blok pamięci (bufor) do składowania (zapisywania) pierwotnego wyglądu (obrazu, *image*) wybranego wycinka prostokątnego ekranu.

W wierszu *22.* widzimy deklarację funkcji własnych klasy `Okno`. Funkcja `VPort` zdefiniowana jest w wierszu *85.* i następnych. Odpowiada ona za wypełnienie tła (za pomocą funkcji bibliotecznej `bar`) kolorem `coVP`. Ponadto, w wierszu *89.*, funkcja ta rysuje osie współrzędnych okna lokalnego (używając funkcji bibliotecznej `line`). Zwróćmy uwagę na jawne konwersje typów `double` na typy `int` (wymagane przez funkcję `line`) bezpośrednio na liście argumentów aktualnych w wywołaniu tej funkcji w wierszu *89*. Dzięki funkcji `VPort`, na ekranie, w trybie graficznym, pojawia się napis `Kliknij punkt`, podpowiadający co należy uczynić po uruchomieniu tego programu.

Funkcja `Kresl` (wiersz *22.*) zdefiniowana jest w wierszu *101.* i następnych. W wierszu *102.* funkcja ta definiuje i uruchamia wskaźnik `p` obiektu strumieniowego `FILE` obsługującego dostęp do pliku `mojPlik`. Wskaźnikowi temu przypisywany jest adres zwracany przez funkcję `fopen`. Gdyby proces otwarcia pliku (za pomocą funkcji `fopen`) się nie powiódł (np. gdyby w bieżącym katalogu nie było pliku *mojPlik*), to wskaźnik `p` otrzymałby wartość `NULL`. Wtedy w wierszu *102.* i następnych wydrukowany zostałby komunikat (jeszcze w trybie graficznym, stąd użycie funkcji `outtextxy`) zawierający napis `mojPlik ?`. Następnie program zostałby zakończony. W przypadku powodzenia otwarcia pliku *mojPlik*, w wierszu *105.* funkcja `rewind(p)` wymusi ustawienie wskaźnika `p` na początku tego pliku. Jest to tutaj zabieg czysto formalny, gdyż funkcja `fopen` automatycznie ustawia wskaźnik na początku otwieranego pliku. Z powyższych uwag wynika, że aby otwarcie pliku się powiodło, należy w bieżącym katalogu zapewnić obecność pliku *mojPlik* np. poprzez wykonanie w tym katalogu programu *p5_6.exe* albo poprzez skopiowanie tego zbioru z katalogu *p5_6*.

W wierszach od *110.* do *114.* widzimy pętlę `do{}while()`. W wierszu *110.* funkcja `fgetc` odczytuje kolejny znak z pliku *mojPlik* i podstawia go do kolejnych elementów tablicy `fY`. W wierszu *112.* funkcja `line` kreśli kolejny odcinek wykresu (od punktu poprzedniego) do węzła, którego rzędna ma odwzorowywać wartość kodu *ASCII* właśnie odczytanego (z pliku) znaku. Każdy węzeł jest kołem wypełnionym. Kreśli go funkcja `pieslice`. Parametry `0,360` (stopni) oznaczają wypełnienie całego koła, a nie tylko jego wycinka. Ostatni parametr, wynoszący `4`, oznacza promień (w pikselach) węzła kołowego. W wierszu *115.* widzimy wywołanie funkcji `pieslice` dla narysowania koła w ostatnim węźle wykresu, bowiem wewnątrz pętli `do{}while()` rysowane są koła w węźle początkowym kolejnego odcinka linii łamanej.

W wierszu *116.* i dwóch następnych widzimy definicje kwadratów (dla każdego węzła) zapewniających łatwą identyfikację danego węzła przez mysz. Zauważmy dodanie offsetów `L` i `T` w celu uzyskania wartości (bezwzględnych) globalnych, gdyż takimi wartościami operują funkcje obsługujące mysz.

W wierszu *23.* widzimy deklarację konstruktora klasy `Okno`. W wierszu *27.* widzimy definicję klasy `Mysz`, podobną do tej z programu *p9_2.cpp*. W wierszu *35.* kreowany jest obiekt `M` (typu `Mysz`) o zasięgu globalnym — dostępny z każdej funkcji.

W wierszu *41.*, w `main`, widzimy wywołanie funkcji `GWlinkowana` uruchamiającej tryb graficzny. W wierszu *43.* kreowany jest obiekt `A` klasy `Okno` poprzez wywołanie na jego rzecz konstruktora tej klasy.

W wierszu *44.* wywołujemy funkcję `VPort` (wiersze *22.* i *85.*), w wyniku czego uzyskujemy współrzędne układu lokalnego. W wierszu *45.* wywołujemy funkcję `Kresl` na rzecz obiektu `A` (klasy `Okno`).

W wierszu *46.* rozpoczyna się zasadnicza część programu w pętli nieskończonej `while(1)`. W wierszu *48.* kontrolowane jest ewentualne naciśnięcie klawisza *Esc* (kod *ASCII* = 27). Po naciśnięciu tego klawisza zostanie wykonana instrukcja `break`. Program przerwie pętlę i przejdzie do wiersza *71.*.

W wierszach *50.* i *51.* (w pętli `for`) sprawdzamy, czy kursor myszy znajduje się w środku któregoś z dziesięciu węzłów i czy równocześnie naciśnięto lewy przycisk myszy. Jeśli tak, to wykonywane jest ciało tej instrukcji w wierszach od *53.* do *71*. Ponadto zmienna `wPam` ustalana jest na zero, gdyż w buforze pamięci wskazywanym przez wskaźnik `pIm` nie ma jeszcze umieszczonego żadnego wycinka obrazu ekranu.

W wierszu *54.* rozpoczyna się pętla `do{}while()` kończąca się w wierszu *66*. Jest ona wykonywana tak długo, jak długo pozostaje naciśnięty przycisk myszy. W wierszu *54.* sprawdzamy, czy wycinek ekranu został już wpisany do bufora w pamięci. Jeśli jeszcze nie (dla właśnie wskazywanego węzła), to wykonywane jest ciało tej instrukcji od wiersza *55.* do wiersza *63*. W zakresie tego ciała, w wierszu *57.*, używamy funkcji `sprintf` do sformatowania i zeskładowania (zapisania) (w tablicy `A.s`) wydruku będącego opisem węzła.

W wierszu *58.* ustalane są współrzędne miejsca, w którym ma zostać wydrukowany opis węzła. Wycinek ekranu, który ma być zmieniony z racji nowego wydruku, jest wcześniej pobrany z ekranu i umieszczony w buforze (przygotowanym w wierszu *84.*). Odbywa się to za pomocą funkcji bibliotecznej `getimage` w wierszu *61*. Wycinek ekranu (którego obraz jest już umieszczony w buforze) wypełniamy w wierszu *62.* prostokątem (`bar`) w kolorze `YELLOW`, ustalonym w wierszu *55.*. Na ten prostokąt, za pomocą funkcji `outtextxy`, nadpisywany jest opis węzła.

Po zwolnieniu przycisku myszy przerwaniu ulega pętla `do{}while()`. Wtedy w wierszu *69.* funkcja `putimage` odtworzy prostokąt za pomocą obrazu ekranu umieszczonego w buforze, na który wskazuje wskaźnik `pIm`.

Naciśnięcie klawisza *Esc* powoduje przerwanie nieskończonej pętli `while(1)` z wiersza *46.* i przejście programu do wiersza *72*. Zamykamy tryb graficzny i zwalniamy pamięć dynamiczną (*heap*), zarezerwowaną dla buforu wycinka obrazu ekranu. Ponieważ jest to tablica obiektów typu `void`, więc do dealokacji pamięci używamy operatora `delete[]`.

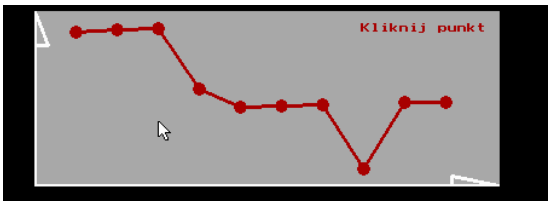
W wierszach od *75.* do *85.* widzimy definicję konstruktora klasy `Okno`. W wierszu *81.* i następnym wartości względne współrzędnych punktów do wykreślenia osi `Y` i `X` lokalnego układu odniesienia zamieniane są na wartości bezwzględne (w pikselach) lokalne.

W wierszu *82.* dokonujemy wpisu sformatowanego napisu wzorcowego złożonego ze znaku i trzycyfrowej liczby; późniejsze wydruki opisów węzłów nie przekroczą rozmiarów tego łańcucha. W wierszu *83.* funkcje `textwidth` i `textheight` zwracają szerokość i wysokość tego wydruku. Na tej podstawie funkcja `imagesize` (w wierszu *84.*) zwraca rozmiar pamięci potrzebnej na bufor do przechowywania obrazu prostokątnego wycinka ekranu o rozmiarze takim, jak rozmiar napisu wzorcowego. Rozmiar bufora podawany jest w bajtach, stąd typ `void` dla bufora. W wierszu *84.* operator `new` dokona rezerwacji pamięci na bufor obrazu wycinka ekranu. Ewentualne niepowodzenie przydziału pamięci dynamicznej spowoduje wykonanie instrukcji z wiersza *85.* i następnego.

Funkcja własna `iniM` z wiersza *122.* inicjalizuje mysz (jak w programie *p9_2.cpp*). Funkcja własna `gdzM` z wiersza *128.* zwraca informację o aktualnym położeniu kursora myszy (`x`, `y`) oraz o stanie naciśnięcia lewego przycisku (`Bu`).

Po skompilowaniu, konsolidacji i wykonaniu programu w trybie *Windows* lub *MS-DOS* otrzymujemy ekran widoczny na rysunku 9.3a.

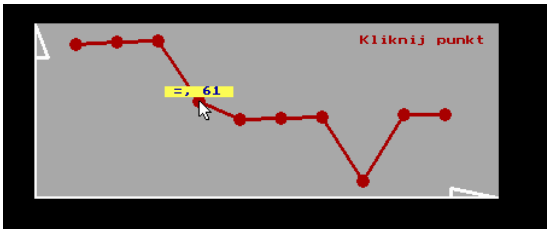
Rysunek 9.3a. *Fragment ekranu programu p9_3.exe z wykresem utworzonym na podstawie pliku mojPlik (z programu p5_6)*



Program *p5_6.exe* wpisał do pliku *mojPlik* ciąg znaków: a, b, c, =, 1, 2, 3, ▯, 4, 4, którym odpowiadają kody *ASCII* wynoszące: 97, 98, 99, 61, 49, 50, 51, 10, 52, 52. Rzędne kolejnych węzłów na powyższym rysunku odpowiadają wartościom powyższych kodów *ASCII* tych znaków.

Jeśli wskazać kursorem myszy jeden z węzłów i nacisnąć lewy przycisk, to nad wskazanym węzłem pojawi się jego opis złożony ze znaku, przecinka oraz wartości kodu *ASCII* tego znaku, co widać na rysunku 9.3b.

Rysunek 9.3b. *Fragment ekranu programu p9_3.exe z informacją dotyczącą znaku (i jego kodu ASCII) ukrytego we wskazanym za pomocą myszy punkcie wykresu*



Mysz wskazuje węzeł czwarty, stąd opis zawiera znak równości o wartości kodu *ASCII* wynoszącej 61. W celu zakończenia programu należy nacisnąć klawisz *Esc*.