Ćwiczenie

„Obsługa klawiatury i myszy”

Tematy ćwiczenia

- klawiatura,
- mysz,
- tworzenie okna konsolowego.

Sprawozdanie

Na każdym ćwiczeniu sporządza się sprawozdanie na bazie materiałów ćwiczenia. Bazowa zawartość sprawozdania musi być przygotowana w domu przed ćwiczeniem (sprawozdanie do ćwiczenia pierwszego jest przygotowywane w czasie ćwiczenia). W czasie ćwiczenia do sprawozdania są dodawane wyniki testowania.

Treść sprawozdania:

strona tytułowa,

spis treści sporządzony za pomocą *Word'*a,

dla każdego punktu rozdziały "Zadanie ", "Opracowanie zadania" (rozdział z tekstem programu i komentarzami), "Testowanie" (rozdział z opisem danych wejściowych i wynikami testowania, w tym zrzuty aktywnego okna).

Nazwa (bez polskich liter, żeby łatwo archiwizować) pliku ze sprawozdaniem musi zawierać nazwę przedmiotu, numer ćwiczenia i nazwisko studenta, na przykład "PN_...".

Plik ze sprawozdaniem musi być przekazany do archiwum grupy.

a) Tworzenie okna konsolowego

Zadanie

Napisać program, w którym otworzyć nowe okno konsolowe.

Wyświetlić okno i komunikaty innymi kolorami niż kolory okna bazowego.

Na środku okna wypisać komunikat „Kolory są ustawione przez ...[1]”.

Ponieważ po uruchomieniu aplikacji okno szybko zniknie, dodać do programu cykl – opóźnienie:

```
      mov   ECX, 07FFFFFFh
etyk: loop  etyk  ; opóźnienie zamknięcia okna
```

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS>>

---

b) Klawiatura

Zadanie

Dodać do programu reakcję na naciśniecie przycisku.

Przyjmować znaki z klawiatury, analizować i wychodzić z aplikacji (zamykać okno) w przypadku kolejnego naciśnięcia przycisków ...[2].

Na dole okna wypisać komunikat o tym, jak zakończyć aplikację.

Cykl – opóźnienie zablokować.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS>>

---

[1] swoje imię i nazwisko

[2] wskazanych w tabeli wariantów

---

c) Mysz

Zadanie

Dodać do programu reakcję na naciśniecie okna myszą.
Jednokrotne naciśniecie lewym przyciskiem myszy musi przemieszczać kursor.
Od miejsca kursora muszą być wyświetlane wprowadzane znaki.
Wychodzić z aplikacji (zamykać okno) po kombinacji naciśnięć przycisku i myszy według reguły ...[3].
Do komunikatu na dole okna dodać informację o tym, jak zakończyć aplikację za pomocą myszy.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS>>

Tabela wariantów

Np	Klawiatura	Mysz		Np	Klawiatura	Mysz
1	A, B	Lewy Alt + prawy myszy		**17**	Q, R	Lewy Ctrl + prawy myszy
2	B, C	Lewy Alt + prawy myszy		**18**	R, S	Lewy Ctrl + prawy myszy
3	C, D	Lewy Alt + prawy myszy		**19**	S, T	Lewy Ctrl + prawy myszy
4	D, E	Lewy Alt + prawy myszy		**20**	T, U	Lewy Ctrl + prawy myszy
5	E, F	Lewy Alt + lewy myszy		**21**	U, V	Lewy Ctrl + lewy myszy
6	F, G	Lewy Alt + lewy myszy		**22**	V, W	Lewy Ctrl + lewy myszy
7	G, H	Lewy Alt + lewy myszy		**23**	W, X	Lewy Ctrl + lewy myszy
8	H, I	Lewy Alt + lewy myszy		**24**	X, Y	Lewy Ctrl + lewy myszy
9	I, J	Prawy Alt + prawy myszy		**25**	Y, Z	Prawy Ctrl + prawy myszy
10	J, K	Prawy Alt + prawy myszy		**26**	Z, A	Prawy Ctrl + prawy myszy
11	K, L	Prawy Alt + prawy myszy		**27**	1, B	Prawy Ctrl + prawy myszy
12	L, M	Prawy Alt + prawy myszy		**28**	2, C	Prawy Ctrl + prawy myszy
13	M, N	Prawy Alt + lewy myszy		**29**	3, D	Prawy Ctrl + lewy myszy
14	N, O	Prawy Alt + lewy myszy		**30**	4, E	Prawy Ctrl + lewy myszy
15	O, P	Prawy Alt + lewy myszy		**31**	5, F	Prawy Ctrl + lewy myszy
16	P, Q	Prawy Alt + lewy myszy		**32**	6, G	Prawy Ctrl + lewy myszy

Wskazówki

System operacyjny daje możliwość otrzymać informacje o stanie klawiatury, myszy, okna i menu.
Każda zmiana stanu sprzętu jest interpretowana jako zdarzenie.
Przy zmianie stanu informacja o zdarzeniu jest zapisywana do 20-bajtowej struktury INPUT_RECORD.
Łańcuchy zapisów o zdarzeniach (ang. *event records*) można czytać za pomocą funkcji ReadConsoleInput.
Struktura INPUT_RECORD ma w pierwszym podwójnym słowie identyfikator źródła zdarzenia. Identyfikator może mieć następujące wartości:
1h = KEY_EVENT – zapis dotyczy klawiatury,
2h = MOUSE_EVENT – zapis dotyczy myszy,
4h = WINDOW_BUFFER_SIZE_EVENT – zapis zawiera informację o zmianie rozmiaru okna,
8h = MENU_EVENT – zapis dotyczy menu,
10h = FOCUS_EVENT – zapis zawiera informację o zmianie fokusu.
W zależności od identyfikatora różne interpretowane są pozostałe 16 bajtów.

[3] z tabeli wariantów

Struktura zapisu typu KEY_EVENT:
4 bajty – znacznik naciśnięcia któregokolwiek klawisza (0 – nie naciśnięty, nie 0 – naciśnięty),
2 bajty – ilość powtórzeń kodu w wyniku długiego naciśnięcia,
2 bajty – wirtualny kod klawisza,
2 bajty – scan-kod klawisza,
2 bajty – ASCII lub UNICODE kod klawisza (funkcja ReadConsoleInputA podaje kod ASCII, a funkcja ReadConsoleInputW podaje kod UNICODE),
4 bajty – podwójne słowo znaczników klawiszy sterujących:
1h = RIGHT_ALT_PRESSED – Alt prawy,
2h = LEFT_ALT_PRESSED – Alt lewy,
4h = RIGHT_CTRL_PRESSED – Ctrl prawy,
8h = LEFT_CTRL_PRESSED – Ctrl lewy,
10h = SHIFT_PRESSED – Shift naciśnięty,
20h = NUMLOCK_ON – Num Lock włączony,
40h = SCROLLLOCK_ON – Scroll Lock włączony,
80h = CAPSLOCK_ON – Caps Lock włączony,
100h = ENHANCED_KEY – klawisz jest zwolniony,

Uwaga: Do rozpoznania, który ze znaczników był naciśnięty, należy stosować rozkaz "test", a nie "cmp", ponieważ może być zaznaczone więcej niż jeden znacznik.

System Windows wysyła do konsoli nie wszystkie kombinacji klawisz, na przykład, nie jest wysyłana kombinacja Ctrl+C. Kombinacja klawisz Ctrl+C daje możliwość zakończyć aplikację awaryjnie.

Struktura zapisu typu MOUSE_EVENT:
2 bajty - współrzędna X kursora myszy,
2 bajty - współrzędna Y kursora myszy,
4 bajty – podwójne słowo znaczników naciśnięcia przycisków myszy:
1h = FROM_LEFT_1ST_BUTTON_PRESSED–przycisk lewy,
2h = RIGHTMOST_BUTTON_PRESSED – przycisk prawy,
4h = FROM_LEFT_2ND_BUTTON_PRESSED – przycisk środkowy (drugi z lewa),
8h = FROM_LEFT_3ND_BUTTON_PRESSED – przycisk trzeci z lewa,
10h = FROM_LEFT_4ND_BUTTON_PRESSED – przycisk czwarty z lewa,

Uwaga: Do rozpoznania, który ze znaczników był naciśnięty, należy stosować rozkaz "test", a nie "cmp", ponieważ może być zaznaczone więcej niż jeden znacznik.

4 bajty – podwójne słowo znaczników naciśnięcia klawiszy sterujących (słowo jest podobne do znaczników struktury KEY_EVENT),
4 bajty – podwójne słowo znaczników przesuwania i podwójnego klikniecie myszy:
1h = MOUSE_MOVED – mysz przesuwa się,
2h = DOUBLE_CLICK - podwójne klikniecie,
4h = MOUSE_WHEELED – poruszenie kółkiem myszy ,

Uwaga: Do rozpoznania, który ze znaczników był naciśnięty, należy stosować rozkaz "test", a nie "cmp", ponieważ może być zaznaczone więcej niż jeden znacznik.

Struktura zapisu typu WINDOW_BUFFER_SIZE_EVENT:
2 bajty – nowy rozmiar okna wzdłuż X,
2 bajty - nowy rozmiar okna wzdłuż Y.

Struktura zapisu typu MENU_EVENT:
4 bajty – identyfikator punktu menu.

Komunikat o zdarzeniu MENU_EVENT jest wykorzystywany systemem operacyjnym i w programie musi być zignorowany.

Struktura zapisu typu FOCUS_EVENT:
4 bajty – znacznik ustawienia fokusu.

Komunikat o zdarzeniu FOCUS_EVENT jest przeznaczony dla systemu operacyjnego i w programie musi być zignorowany.

Deklaracje struktur

Funkcje API Win32, które są potrzebne do realizacji operacji z klawiatura i myszą, używają następujących struktur:

```
COORD STRUCT     ; struktura z współrzędnymi pozycji kursora
      X     DW    0     ; współrzędna X
      Y     DW    0     ; współrzędna Y
COORD ENDS

MOUSE_EVENT_RECORD STRUCT
  dwMousePosition   COORD <> ; współrzędne X, Y kursora myszy
  dwButtonState     DWORD ? ; znaczniki naciśnięcia przycisków myszy
  dwControlKeyState DWORD ? ; znaczniki naciśnięcia klawiszy sterujących
  dwEventFlags      DWORD ? ; znaczniki przesuwania i podwójnego klikniecie
myszy
MOUSE_EVENT_RECORD ENDS

KEY_EVENT_RECORD STRUCT
  bKeyDown      DWORD ? ;znacznik naciśnięcia któregokolwiek klawisza
  wRepeatCount  WORD ?  ; ilość powtórzeń kodu przy długim naciśnięciu
  wVirtualKeyCode   WORD ? ; wirtualny kod klawisza
  wVirtualScanCode  WORD ? ; scan-kod klawisza
  UNION
    UnicodeChar     WORD ? ; UNICODE kod klawisza
    AsciiChar       BYTE ? ; ASCII kod klawisza
  ENDS
  dwControlKeyState DWORD ? ;znaczniki klawiszy sterujących
KEY_EVENT_RECORD ENDS

WINDOW_BUFFER_SIZE_RECORD STRUCT
  dwSize  COORD <>
WINDOW_BUFFER_SIZE_RECORD ENDS

MENU_EVENT_RECORD STRUCT
  dwCommandId  DWORD      ?
MENU_EVENT_RECORD ENDS

FOCUS_EVENT_RECORD STRUCT
  bSetFocus  DWORD      ?
FOCUS_EVENT_RECORD ENDS

INPUT_RECORD STRUCT  ;struktura z informacją o zdarzeniu
  EventType             WORD ? ; typ zdarzenia
  two_byte_alignment    WORD ? ; wyrównanie do granicy/4
  UNION
    KeyEvent                KEY_EVENT_RECORD            <>
    MouseEvent              MOUSE_EVENT_RECORD          <>
    WindowBufferSizeEvent   WINDOW_BUFFER_SIZE_RECORD   <>
    MenuEvent               MENU_EVENT_RECORD           <>
    FocusEvent              FOCUS_EVENT_RECORD          <>
  ENDS
INPUT_RECORD ENDS
```

Zmienna typu struktura

Zmienną typu struktura definiujemy w sekcji danych, na przykład

```
.DATA
YX          COORD      <>
zapis       INPUT_RECORD <>
```

W odwołaniu do pola zmiennej używamy „kropkę”, a przy odwołaniu do zmiennej struktury stosujemy operator PTR w celu rzutowania typu.

Przykładowo:

```
.CODE
mov YX.X,0
mov YX.Y,0
INVOKE FillConsoleOutputAttribute,...,...,...,COORD PTR [YX],...
```

Jeśli pole struktury jest złożonym typem danych, należy stosować etapowe przekształcenie typów. Przykładowo:

```
.DATA
hout       DD    ?     ; deskryptor buforu wyjściowego
hinp       DD    ?     ; deskryptor buforu wejściowego
nzdarz     DD    ?     ; ilość zdarzeń
YX         COORD <>
zapis      INPUT_RECORD <>
.CODE
;.....
;--- pętla
powt:
;--- odczyt komunikatu zdarzenia
 INVOKE FlushConsoleInputBuffer,hinp ;czyszczenie kolejki komunikatów
 INVOKE ReadConsoleInputA,hinp,OFFSET zapis,1,OFFSET nzdarz ; odczyt
komunikatu zdarzenia
;--- sprawdzanie typu zdarzenia
 cmp  WORD PTR [zapis.EventType], MOUSE_EVENT
 je   mysz
 cmp  WORD PTR [zapis.EventType], KEY_EVENT
 je   @F
 jmp  powt
@@:
 jmp  klaw
;--- komunikat od myszy
mysz:
 test (MOUSE_EVENT_RECORD PTR [zapis.MouseEvent]).dwEventFlags,
DOUBLE_CLICK
 jz  @F
 jmp  kon       ; na koniec programu
@@:
 test (MOUSE_EVENT_RECORD PTR [zapis.MouseEvent]).dwButtonState,
FROM_LEFT_1ST_BUTTON_PRESSED
 jnz   @F
 jmp  powt
@@:
;--- jest naciśnięty lewy klawisz myszy
 mov  EAX, (MOUSE_EVENT_RECORD PTR [zapis.MouseEvent]).dwMousePosition
 mov  COORD PTR [YX], EAX   ; współrzędne X,Y
 INVOKE SetConsoleCursorPosition,hout,COORD PTR [YX] ; ustawienie pozycji
kursora
 jmp  powt
;--- komunikat od klawiatury ---
klaw:
 cmp  (KEY_EVENT_RECORD PTR [zapis.KeyEvent]).bKeyDown, 0   ; porównanie z
zerem
 jne  @F
 jmp powt  ;jeśli nie ma naciśnięcia
@@:
 mov AL,(KEY_EVENT_RECORD PTR [zapis.KeyEvent]).ASCIIChar
 mov symb,AL    ; zapisywanie znaku
 cmp  AL, 041h   ; porównanie z kodem a
 jne  @F
 jmp  etyk
```

```
@@:
 cmp  AL, 061h   ; porównanie z kodem A
 jne  @F
 jmp  etyk
@@:
 INVOKE WriteConsoleOutputCharacterA,hout,OFFSET symb,1,
COORD PTR [YX],OFFSET rout  ; wypisywanie znaku
 jmp  powt
etyk:
 ....
 jne @F
 jmp  kon
@@:
 jmp powt  ; na powtórzenie
```

Kolory w aplikacjach konsolowych

Kolory w aplikacjach konsolowych zakodowane są następująco:

bity z lewej strony na prawo:

BG_I (intensywność tła), BG_R (czerwony tła), BG_G (zielony tła), BG_B (niebieski tła),
FG_I (intensywność znaku), FG_R (czerwony znaku), FG_G (zielony znaku), FG_B (niebieski znaku)

A więc można ustawić 16 kolorów tła i 16 kolorów znaku.

Przykładowy kolor 11111100b to białe tło i jasnoczerwony znak.

Napełnienie komórek jednakowym atrybutem

```
;--- obliczenie rozmiaru buforu ekranu
 INVOKE GetLargestConsoleWindowSize,hout
 mov  YX, EAX
 movzx     EAX, YX.X
 movzx     EBX, YX.Y
 mul  EBX
 mov  rkom, EAX
 ;--- napełnienie komórek jednakowym atrybutem ---
 mov  YX.X, 0
 mov  YX.Y, 0
 INVOKE FillConsoleOutputAttribute,hout,atryb,rkom, \
             COORD PTR [YX],OFFSET rfakt
```

Ustawienie pozycji kursora

```
 mov YX.X, 0
 mov  YX.Y, 1
 INVOKE  SetConsoleCursorPosition,hout,  COORD  PTR  [YX]  ;ustawienie  pozycji
kursora
```

Wyprowadzenie komunikatu informacyjnego

```
 INVOKE SetConsoleTextAttribute,hout,atryb
 INVOKE CharToOemA,OFFSET wiersz,OFFSET wiersz; konwersja polskich znaków
 INVOKE WriteConsoleA,hout,OFFSET wiersz, rwiersz, OFFSET rout,0
```

Funkcje API Win32, które są związane z konsolą, klawiaturą i myszą

AllocConsole

Alokacja konsoli

FreeConsole

Zamykanie wszystkich konsoli

SetConsoleTitleA

Wypisywanie nagłówka okna konsolowego.

Argument:

lpConsoleTitle – adres wiersza - nagłówka

SetConsoleTextAttribute

Ustawienie atrybutów tekstowych.

Argumenty:

hConsoleOutput – deskryptor buforu wyjściowego konsoli

wAttributes – kolor (typu WORD)

GetLargestConsoleWindowSize

Zwraca maksymalny rozmiar (typ COORD) okna konsoli.

Argument:

hConsoleOutput – deskryptor buforu wyjściowego konsoli

FillConsoleOutputAttribute

Wpisuje do buforu wyjściowego (ekranu) atrybuty znakowe.

Argumenty:

hConsoleOutput – deskryptor buforu wyjściowego konsoli,

wAttribute – kolor (typu WORD),

nLength – ilość pozycji,

dwWriteCoord – współrzędne X, Y (typ COORD) pierwszej pozycji w oknie konsoli,

lpNumberOfAttrsWritten – adres zmiennej, do której będzie zapisana faktyczna ilość pozycji.

SetConsoleCursorPosition

Ustawienie pozycji kursora.

Argumenty:

hConsoleOutput – deskryptor buforu wyjściowego konsoli,

dwCursorPosition – współrzędne X, Y (typ COORD) pozycji w oknie konsoli.

ReadConsoleInputA

Czyta komunikaty okna konsoli.

Argumenty:

hConsoleInput – deskryptor buforu wejściowego konsoli,

lpBuffer – adres obiektu typu INPUT_RECORD,

nLength – ilość komunikatów (zapisów),

lpNumberOfEventsRead – adres zmiennej, do której będzie zapisana faktyczna ilość przeczytanych komunikatów (zapisów).

FlushConsoleInputBuffer

Zerowanie kolejki komunikatów.

Argument:

hConsoleInput – deskryptor buforu wejściowego konsoli.

WriteConsoleOutputCharacterA

Wyprowadzenie znaków od pozycji kursora.

Argumenty:

hConsoleOutput – deskryptor buforu wyjściowego konsoli,

lpCharacter – adres bufora z tekstem,

nLength – ilość znaków,

dwWriteCoord – współrzędne X, Y (typ COORD) pierwszej pozycji w oknie konsoli,

lpNumberOfCharsWritten – adres zmiennej, do której będzie zapisana faktyczna ilość wyprowadzonych znaków.

WriteConsoleA

Wyprowadzenie znaków do buforu konsoli

Argumenty:

hConsoleOutput – deskryptor buforu wyjściowego konsoli,

lpBuffer – adres bufora z tekstem,

nNumberOfCharsToWrite – ilość znaków,

lpNumberOfCharsWritten – adres zmiennej, do której będzie zapisana faktyczna ilość wyprowadzonych znaków,.

lpReserved – rezerwa (musi być 0).

Fragmenty programu

```
;--- sekcja kodu ---------
_TEXT SEGMENT
start:
;...........
;--- tworzenie konsoli ---
 INVOKE     FreeConsole
 INVOKE     AllocConsole
;............
;--- obliczenie rozmiaru buforu ekranu
 INVOKE GetLargestConsoleWindowSize,hout
 mov   YX, EAX
 movzx EAX, YX.X
 movzx EBX, YX.Y
 mul   EBX
 mov   rkom, EAX
;--- napełnienie komórek jednakowym atrybutem ---
 mov   YX.X, 0
 mov   YX.Y, 0
 INVOKE FillConsoleOutputAttribute,hout,atryb,rkom,
COORD PTR [YX],OFFSET rfakt
;--- ustawienie koloru ekrany ---
 INVOKE SetConsoleTextAttribute,hout,atryb
;..............
;--- zamknięcie konsoli
kon:
 INVOKE     FreeConsole
;----- zakończenie aplikacji ---------
 INVOKE     ExitProcess,0
_TEXT  ENDS
END start
```

Program przykładowy

```
.586P
.MODEL flat, STDCALL
;--- z pliku windows.inc ---
STD_INPUT_HANDLE              equ -10
STD_OUTPUT_HANDLE               equ -11

KEY_EVENT    EQU 1h ; zdarzenie klawiatury
MOUSE_EVENT         EQU 2h ; zdarzenie myszy
MENU_EVENT  EQU 8h
FOCUS_EVENT EQU 10h

RIGHT_ALT_PRESSEDEQU    1h ; Alt prawy
LEFT_ALT_PRESSED  EQU    2h ; Alt lewy
RIGHT_CTRL_PRESSED        EQU    4h ; Ctrl prawy
LEFT_CTRL_PRESSEDEQU    8h ; Ctrl lewy
SHIFT_PRESSED             EQU    10h ; Shift naciśnięty
NUMLOCK_ON                EQU    20h ; Num Lock włączony
SCROLLLOCK_ON             EQU    40h ; Scroll Lock włączony
CAPSLOCK_ON               EQU    80h ; Caps Lock włączony
ENHANCED_KEY              EQU    100h ; klawisz jest zwolniony
FROM_LEFT_1ST_BUTTON_PRESSED         EQU    1h ; przycisk lewy
RIGHTMOST_BUTTON_PRESSED     EQU    2h ; przycisk prawy
FROM_LEFT_2ND_BUTTON_PRESSED EQU 4h ; przycisk srodkowy
FROM_LEFT_3ND_BUTTON_PRESSED EQU 8h ; przycisk trzeci z lewa
```

```
FROM_LEFT_4ND_BUTTON_PRESSED EQU 10h ;przycisk 4-ty z lewa
MOUSE_MOVED         EQU    1h ; mysz przesuwa się
DOUBLE_CLICK        EQU    2h ; podwójne klikniecie
MOUSE_WHEELED       EQU    4h ; poruszenie kółkiem myszy

;--- struktury ----------
COORD  STRUCT
  X         DW      ?
  Y         DW      ?
COORD  ENDS

MOUSE_EVENT_RECORD STRUCT
  dwMousePosition   COORD <> ; współrzędne X, Y kursora myszy
  dwButtonState     DWORD ? ; znaczniki naciśnięcia przycisków myszy
  dwControlKeyState DWORD ? ; znaczniki naciśnięcia klawiszy sterujących
  dwEventFlags      DWORD ? ; znaczniki przesuwania i podwójnego klik-niecie myszy
MOUSE_EVENT_RECORD ENDS

KEY_EVENT_RECORD STRUCT
     bKeyDown DWORD ?         ; znacznik naciśnięcia któregokolwiek klawisza
     wRepeatCount WORD ? ; ilość powtórzeń kodu przy długim naciśnięciu
     wVirtualKeyCode WORD ? ; wirtualny kod klawisza
     wVirtualScanCode WORD ? ; scan-kod klawisza
     UNION
          UnicodeChar WORD ?     ; UNICODE kod klawisza
          AsciiChar BYTE ?                ; ASCII kod klawisza
     ENDS
     dwControlKeyState DWORD ?     ; znaczniki klawiszy sterujących
KEY_EVENT_RECORD ENDS

WINDOW_BUFFER_SIZE_RECORD STRUCT
  dwSize  COORD <>
WINDOW_BUFFER_SIZE_RECORD ENDS

MENU_EVENT_RECORD STRUCT
  dwCommandId  DWORD     ?
MENU_EVENT_RECORD ENDS

FOCUS_EVENT_RECORD STRUCT
  bSetFocus  DWORD     ?
FOCUS_EVENT_RECORD ENDS


INPUT_RECORD STRUCT
     EventType WORD ?
     two_byte_alignment WORD ?
    UNION
     KeyEvent KEY_EVENT_RECORD <>
     MouseEvent            MOUSE_EVENT_RECORD         <>
     WindowBufferSizeEvent   WINDOW_BUFFER_SIZE_RECORD   <>
     MenuEvent             MENU_EVENT_RECORD          <>
     FocusEvent            FOCUS_EVENT_RECORD         <>
    ENDS
INPUT_RECORD ENDS

;--- funkcje API Win32 z pliku  user32.inc ---
CharToOemA PROTO :DWORD,:DWORD
;--- funkcje API Win32 z pliku kernel32.inc -----------
GetStdHandle PROTO :DWORD
```

```
WriteConsoleA PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
AllocConsole PROTO
FreeConsole PROTO
SetConsoleTitleA PROTO :DWORD
   ;;BOOL SetConsoleTitle(
   ;;LPCTSTR lpConsoleTitle      // address of new title
   ;;);
SetConsoleTextAttribute PROTO :DWORD,:DWORD
   ;;BOOL SetConsoleTextAttribute(
   ;;HANDLE hConsoleOutput,   // handle of console screen buffer
   ;;WORD wAttributes   // text and background colors
   ;;);
GetLargestConsoleWindowSize PROTO :DWORD
   ;;COORD GetLargestConsoleWindowSize(
   ;;HANDLE hConsoleOutput    // handle of console screen buffer
   ;;);
   ;;typedef struct _COORD { // coord.
   ;;SHORT X;      // horizontal coordinate
   ;;SHORT Y;      // vertical coordinate
   ;;} COORD;
FillConsoleOutputAttribute PROTO :DWORD,:DWORD,:DWORD,:COORD,:DWORD
   ;;BOOL FillConsoleOutputAttribute(
   ;;HANDLE hConsoleOutput,   // handle to screen buffer
   ;;WORD wAttribute,   // color attribute to write
   ;;DWORD nLength,    // number of character cells to write to
   ;;COORD dwWriteCoord,       // x- and y-coordinates of first cell
   ;;LPDWORD lpNumberOfAttrsWritten           // pointer to number of cells written to
   ;;);
SetConsoleCursorPosition PROTO :DWORD, :COORD
   ;;BOOL SetConsoleCursorPosition(
   ;;HANDLE hConsoleOutput,   // handle of console screen buffer
   ;;COORD dwCursorPosition    // new cursor position coordinates
   ;;);
ReadConsoleInputA PROTO :DWORD,:DWORD,:DWORD,:DWORD
   ;;BOOL ReadConsoleInput(
   ;;HANDLE hConsoleInput,      // handle of a console input buffer
   ;;PINPUT_RECORD lpBuffer,// address of the buffer for read data
   ;;DWORD nLength,    // number of records to read
   ;;LPDWORD lpNumberOfEventsRead           // address of number of records read
   ;;);
FlushConsoleInputBuffer PROTO :DWORD
   ;;BOOL FlushConsoleInputBuffer(
   ;;HANDLE hConsoleInput       // handle to console input buffer
   ;;);
WriteConsoleOutputCharacterA PROTO :DWORD,:DWORD,:DWORD,:COORD,:DWORD
   ;;BOOL WriteConsoleOutputCharacter(
   ;;HANDLE hConsoleOutput,   // handle to a console screen buffer
   ;;LPCTSTR lpCharacter,         // pointer to buffer to write characters from
   ;;DWORD nLength,    // number of character cells to write to
   ;;COORD dwWriteCoord,       // coordinates of first cell to write to
   ;;LPDWORD lpNumberOfCharsWritten          // pointer to number of cells written to
  ;;);
ExitProcess PROTO :DWORD
   ;;VOID ExitProcess(
   ;;UINT uExitCode       // exit code for all threads
   ;;);
wsprintfA PROTO C :VARARG

;--- biblioteki ---------
```

```
includelib ..\lib\user32.lib
includelib ..\lib\kernel32.lib
includelib ..\lib\masm32.lib

;--- Kolory ----
kolor1 = 11111100b        ; Kolory: BG_I,R,G,B,FG__I,R,G,B
kolor2 = 11111001b        ; Kolory: BG_I,R,G,B,FG__I,R,G,B

;--- sekcja danych ------
_DATA SEGMENT



_DATA ENDS
;--- sekcja kodu ---------
_TEXT SEGMENT
start:
;--- tworzenie konsoli ---
 INVOKE          FreeConsole
 INVOKE          AllocConsole



;--- zamknięcie konsoli
kon:
 INVOKE          FreeConsole
;----- wywołanie funkcji ExitProcess ---------
 INVOKE          ExitProcess,0
_TEXT   ENDS
END start
```