1. WYBRANE FUNKCJE OGÓLNE I PLIKI DYSKOWE

1.1. Pomiar czasu wykonania programu <windows.h>

DWORD **GetTickCount**(void)

Funkcja zwraca liczbę ***milisekund***, jakie upłynęły od momentu startu systemu.

```
DWORD  StartTime = GetTickCount()      // aktualny czas - moment rozpoczęcia
        { WYKONYWANY_KOD_PROGRAMU ; }
DWORD  ElapsedTime = GetTickCount() - StartTime;
```

```
DWORD Start = GetTickCount();
long ile = pow(10, 6);
for(long i=0; i<ile; i++);
cout<<GetTickCount() - Start;
```

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main()                                          // TickCount1
{
DWORD OldTime = GetTickCount();
DWORD ActualTime = 0;
int n = 0;
while(n < 9) {
  ActualTime = GetTickCount();
  if((ActualTime - OldTime) >= 50) {
      n++;
      cout << "Obieg" << n << ":  " << ActualTime << endl;
      OldTime = ActualTime;  }
 }
cout << "\nWcisnij Enter";  // cin.get();
return 0;
}
```

```
Obieg1:  102664578
Obieg2:  102664640
Obieg3:  102664703
Obieg4:  102664765
Obieg5:  102664828
Obieg6:  102664890
Obieg7:  102664953
Obieg8:  102665015
Obieg9:  102665078
```

Program **TickCount2** co 50 ms wyświetla stan licznika milisekund.

Po upływie 4 294 967 295 milisekund (po 49.7 dniach) -od momentu uruchomienia systemu Windows- licznik zostaje wyzerowany a jego zwiększanie rozpoczyna się od nowa.

Systemowy licznik milisekund to 32-bitowa zmienna (DWORD), która umożliwia przechowywanie liczb w przedziale od 0 do 4 294 967 295.

```
#include <iostream>
#include <cmath>
#include <Windows.h>
using namespace std;
int main()                                          // TickCount2
{
DWORD OldTime, ActualTime = 0, Time = 0;
OldTime = GetTickCount();
for (long k=0; k < 99000; k++) log(pow((pow(sin(k)+1.1 ,3.3)), 2.2));
ActualTime = GetTickCount();
Time = ActualTime - OldTime;
cout << Time<< endl;
cout << "\nWcisnij Enter";
// cin.get();
return 0;
}
```


1.2. Pomiar czasu wykonania wątku

Funkcja **GetThreadTimes** zwraca ilość czasu CPU wykorzystanego przez wątek.

```
BOOL GetThreadTimes( HANDLE     hThread,          // uchwyt wątku
                     LPFILETIME lpCreationTime,   // czas utworzenia
                     LPFILETIME lpExitTime,       // czas wyjścia
                     LPFILETIME lpKernelTime,     // czas jądra
                     LPFILETIME lpUserTime        // czas użytkownika
                   )
```

lpCreationTime – wskaźnik na strukturę **FILETIME**, zawierającą czas utworzenia wątku.

lpExitTime - wskaźnik na strukturę **FILETIME**, zawierającą czas usunięcia wątku.
Jeśli wątek nadal działa, czas wyjścia jest niezdefiniowany.

Thread creation and exit times are points in time expressed as the amount of time that has elapsed since midnight on January 1, 1601 at Greenwich, England.

lpKernelTime -wskaźnik na strukturę **FILETIME**, zawierającą czas zużyty przez wątek na wykonywanie kodu systemu operacyjnego.

lpUserTime -wskaźnik na strukturę **FILETIME**, zawierającą czas zużyty przez wątek na wykonanie kodu aplikacji.

Thread kernel mode and user mode times are amounts of time.

For example, if a thread has spent one second in kernel mode, this function will fill the **FILETIME** structure specified by *lpKernelTime* with a 64-bit value of ten million.
That is the number of 100-nanosecond units in one second.

1.3. Czas i Data

```
int GetTimeFormat(
  LCID Locale,                  // locale for which time is to be formatted
  DWORD dwFlags,                // flags specifying function options
  CONST SYSTEMTIME *lpTime,     // time to be formatted
  LPCTSTR lpFormat,             // time format string
  LPTSTR lpTimeStr,             // pointer to a buffer that receives the formatted time string.
  int cchTime                   // size, in bytes or characters, of the buffer
 );
```

```
int GetDateFormat(
  LCID Locale,
  DWORD dwFlags,
  CONST SYSTEMTIME *lpDate,
  LPCTSTR lpFormat,
  LPTSTR lpDateStr,
  int cchDate,
 );
```

```
#include <windows.h>
#include <iostream>
using namespace std;
int main()                                          // Czas
{
char buf[256];
  GetTimeFormat(NULL, 0, NULL, NULL, buf, 255);
      cout<<buf<<endl;
  GetDateFormat(NULL, NULL, NULL, NULL, buf, 255);
      cout<<buf<<endl;
cout << "\nWcisnij Enter"; cin.get();
return 0;
}
```

```
23:04:50
2013-10-04
```

VOID **GetSystemTime(**

LPSYSTEMTIME *lpSystemTime* // address of system time structure

);

Function retrieves the current system date and time.

The system time is expressed in Coordinated Universal Time (UTC).

lpSystemTime - points to a SYSTEMTIME structure to receive the current system date and time.

The time copied to the SYSTEMTIME structure may not be the same as the local time - the date and time of day for your time zone

BOOL **SystemTimeToFileTime(**

CONST SYSTEMTIME **lpSystemTime*, // address of system time to convert

LPFILETIME *lpFileTime* // address of buffer for converted file time

);

Function converts a system time to a file time.

lpSystemTime **-**points to a SYSTEMTIME structure that contains the time to be converted.
The wDayOfWeek member of the SYSTEMTIME structure is ignored.

lpFileTime -points to a FILETIME structure to receive the converted system time.

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero.

❒ **SYSTEMTIME** structure represents a date and time using individual members.

```
typedef struct _SYSTEMTIME {    // st
  WORD wYear;
  WORD wMonth;                  // January = 1, February = 2, and so on
  WORD wDayOfWeek;              // Sunday = 0, Monday = 1, and so on
  WORD wDay;
  WORD wHour;
  WORD wMinute;
  WORD wSecond;
  WORD wMilliseconds;
} SYSTEMTIME;
```

It is not recommended that you add and subtract values from the SYSTEMTIME structure to obtain relative times. Instead, you should:

-Convert the SYSTEMTIME structure to a FILETIME structure.

-Copy the resulting FILETIME structure to a LARGE_INTEGER structure.

-Use normal 64-bit arithmetic on the LARGE_INTEGER value.

❒ **FILETIME** structure is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601.

```
typedef struct _FILETIME {    // ft
  DWORD   dwLowDateTime;      // specifies the low-order 32 bits of the file time
  DWORD   dwHighDateTime;     // specifies the high-order 32 bits of the file time
} FILETIME;
```


It is not recommended that you add and subtract values from the FILETIME structure to obtain relative times.

Instead, you should

-Copy the resulting FILETIME structure to a LARGE_INTEGER structure.

-Use normal 64-bit arithmetic on the LARGE_INTEGER value.

❒ **LARGE_INTEGER** structure is used to represent a 64-bit signed integer value.

```
typedef union _LARGE_INTEGER {
  struct {
    DWORD LowPart;      // specifies the low-order 32 bits
    LONG  HighPart;     // specifies the high-order 32 bits
    };
  LONGLONG  QuadPart;   // Specifies a 64-bit signed integer
} LARGE_INTEGER;
```

The LARGE_INTEGER structure is actually a union.

If your compiler has built-in support for 64-bit integers, use the QuadPart member to store the 64-bit integer.

Otherwise, use the LowPart and HighPart members to store the 64-bit integer.

1.4. Funkcje działające na pamięci RAM

```
VOID CopyMemory(    // function copies a block of memory from one location to another
  PVOID         Destination,    / points to the starting address of the copied block's destination
  CONST VOID *  Source,         // points to the starting address of the block of memory to copy.
  DWORD         Length          // specifies the size, in bytes, of the block of memory to copy.
);
```

If the source and destination blocks overlap, the results are undefined

```
VOID FillMemory(    // function fills a block of memory with a specified value
  PVOID         Destination,    // points to the starting address of the block of memory to fill.
  DWORD         Length,         // specifies the size, in bytes, of the block of memory to fill.
  BYTE          Fill            // specifies the byte value with which to fill the memory block.
);
```

```
VOID ZeroMemory(    // function fills a block of memory with zeros
  PVOID   Destination,      // points to the starting address of the block of memory to fill with zeros.
  DWORD   Length            // specifies the size, in bytes, of the block of memory to fill with zeros.
);
```

```
VOID MoveMemory(    // function moves a block of memory from one location to another
    PVOID         Destination,  // points to the starting address of the destination of the move.
    CONST VOID *  Source,       // points to the starting address of the block of memory to move.
    DWORD         Length        // specifies the size, in bytes, of the block of memory to move.
    );
```

The source and destination blocks may overlap.

1.5. Funkcja GetLastError

```
DWORD GetLastError()
```

Funkcja zwraca 32-bitowy kod błędu wykrytego w wątku, który można przetłumaczyć na użyteczną postać.

winerror.h -- error code definitions for the Win32 API functions.
http://www.carrona.org/winerror.html

Po wykryciu błędu wewnątrz funkcji następuje skojarzenie numeru kodu błędu z wątkiem wywołującym za pomocą **lokalnego magazynu wątków** (*thread-local storage)*.

ERROR_INVALID_FUNCTION	1
ERROR_FILE_NOT_FOUND	2
ERROR_PATH_NOT_FOUND	3
ERROR_TOO_MANY_OPEN_FILES	4
ERROR_ACCESS_DENIED	5
ERROR_FILE_EXISTS	80
ERROR_DISK_FULL	112
ERROR_SEM_TIMEOUT	121
ERROR_SEM_NOT_FOUND	187
ERROR_FILE_ENCRYPTED	6002

Wątki mogą działać niezależnie, nie zakłócając sobie nawzajem zwracanych kodów błędów.

Po zakończeniu działania funkcji o wystąpieniu błędu informuje jej wartość zwracana.

Rodzaj napotkanego błędu udostępnia funkcja ***GetLastError***:

➔ ***GetLastError*** zwraca **ostatni błąd wygenerowany przez wątek**.

☛ **Jeśli** wątek wywoła funkcję, która zakończy się pomyślnie, poprzedni kod błędu nie zostanie nadpisany i nie będzie wskazywał powodzenia.
Większość funkcji stosuje się do tej reguły.

Niektóre funkcje mogą kończyć się **powodzeniem** z **wielu** powodów:
-udało utworzyć się taki obiekt,
-obiekt o podanej nazwie już istnieje.

Czasami aplikacja musi znać dokładnie przyczynę powodzenia.

Do przekazania tej informacji wykorzystuje się mechanizm **kodu ostatniego błędu**.

Po **pomyślnym** wykonaniu niektórych funkcji dodatkowe informacje uzyskuje się wywołując ***GetLastError***.

Dokumentacja SDK zawiera informację, czy dana funkcja zachowuje się w taki sposób.

1.6. Funkcje plikowe (Obiekty plików)

Obiekty plików to konstrukcje **trybu jądra**, są zasobami systemowymi i mogą być:
-współdzielone przez wiele procesów trybu użytkownika,
-chronione zabezpieczeniami bazującymi na obiektach oraz obsługiwać synchronizację.

Utworzenie w jądrze **ObiektuPliku** wymaga wywołania funkcji **CreateFile**.

Funkcja może służyć do otwierania plików, strumieni plików, dysków, partycji, konsoli i innych.

```
HANDLE CreateFile(
  LPCTSTR                lpFileName,            // pointer to name of the file  (np.: "COM1")
  DWORD                  dwDesiredAccess,       // access (read-write) mode
  DWORD                  dwShareMode,           // share mode
  LPSECURITY_ATTRIBUTES  lpSecurityAttributes,  // pointer to security attributes (NULL)
  DWORD                  dwCreationDistribution,// how to create
  DWORD                  dwFlagsAndAttributes,  // file attributes
  HANDLE                 hTemplateFile          // handle to file with attributes to copy
);
```

Zwraca **uchwyt** ObiektuPliku, gdy zakończona jest powodzeniem,
w przeciwnym razie wartość błędu: **INVALID_HANDLE_VALUE**

lpFileName: nazwa pliku, który ma być utworzony/otworzony (ścieżka nieobowiązkowa);

dwDesiredAccess: 0 –można jedynie odczytać atrybuty pliku,
GENERIC_READ -plik do czytania,
GENERIC_WRITE -plik do pisania,
GENERIC_READ | GENERIC_WRITE –plik do czytania i pisania.

dwShareMode: definiuje metodę dzielenia się plikiem
0 -dalsze próby otwarcia tego pliku nie powiodą się,
FILE_SHARE_READ
-dalsze próby otwarcia tego pliku z GENERIC_WRITE nie powiodą się,
FILE_SHARE_WRITE
- dalsze próby otwarcia tego pliku z GENERIC_READ nie powiodą się,
FILE_SHARE_READ | FILE_SHARE_WRITE
- dalsze próby otwarcia tego pliku powiodą się.

dwCreationDistribution: **CREATE_NEW, CREATE_ALWAYS**,
OPEN_EXISTING, OPEN_ALWAYS,
TRUNCATE_EXISTING

dwFlagsAndAttributes: FILE_ATTRIBUTE_ARCHIVE,
FILE_ATTRIBUTE_COMPRESSED,
FILE_ATTRIBUTE_HIDDEN,
FILE_ATTRIBUTE_NORMAL,
FILE_ATTRIBUTE_OFFLINE,
FILE_ATTRIBUTE_READONLY,
FILE_ATTRIBUTE_SYSTEM,
FILE_ATTRIBUTE_TEMPORARY.


```
BOOL ReadFile(
  HANDLE       hFile,                  // handle of file to read
  LPVOID       lpBuffer,               // address of buffer that receives data
  DWORD        nNumberOfBytesToRead,   // number of bytes to read
  LPDWORD      lpNumberOfBytesRead,    // address of number of bytes read
  LPOVERLAPPED lpOverlapped            // address of structure for data  (NULL)
);
```

```
BOOL WriteFile(
  HANDLE       hFile,                  // handle to file to write to
  LPCVOID      lpBuffer,               // pointer to data to write to file
  DWORD        nNumberOfBytesToWrite,  // number of bytes to write
  LPDWORD      lpNumberOfBytesWritten, // pointer to number of bytes written
  LPOVERLAPPED / pOverlapped           // pointer to structure needed for overlapped I/O, NULL
);
```

```
DWORD SetFilePointer(
  HANDLE hFile,                 // handle of file
  LONG   lDistanceToMove,       // number of bytes to move file pointer
  PLONG  lpDistanceToMoveHigh,  // address of high-order word of distance to move,  NULL
  DWORD  dwMoveMethod           // how to move:  FILE_BEGIN,  FILE_CURRENT,  FILE_END
);
```

If the function fails, the return value is **0xFFFFFFFF**.

```
DWORD GetFileSize(
  HANDLE  hFile,          // handle of file to get size of
  LPDWORD lpFileSizeHigh  // address of high-order word for file size  (NULL if not require)
);
```

Function retrieves the size, in bytes, of the specified file

If the function fails and *lpFileSizeHigh* is NULL, the return value is **0xFFFFFFFF**.

The handle must have been created with either GENERIC_READ or GENERIC_WRITE access to the file.

Program **Plik1** otwiera ***istniejący*** tekstowy plik dyskowy i dokonuje odczytu jego zawartości korzystając wyłącznie z funkcji systemu operacyjnego Windows.

```
zawartość pliku Plik1.txt
123456789 Tekst tego pliku
zawiera tylko dwa wiersze tekstu.
```

```
#include <windows.h>                    // CreateFile      SetFilePointer      ReadFile
#include <cstdlib>
#include<cstdio>
using namespace std;
#define BYTEStoREAD  150                                  // liczba znaków do odczytania
DWORD INVALID_SET_FILE = 0xFFFFFFFF;
int main()                                                // Plik1
{
DWORD  OFFSET = 7;                      // liczba znaków, które zostaną opuszczone
DWORD  readed = 0;                      // liczba znaków odczytanych
HANDLE hFile;                           // Uchwyt do pliku
char Wy[BYTEStoREAD];                   // bufor Wy
char nameFile[33] = "Plik1.txt";                          // "D:\\Plik1.txt";
hFile = CreateFile(       // ----------utworzenie pliku do odczytu
            nameFile,                   // Nazwa pliku
            GENERIC_READ,               // Tylko czytanie z pliku
            FILE_SHARE_READ,            // Współdzielenie czytania z pliku
            NULL,                       // Standardowe parametry bezpieczeństwa
            OPEN_EXISTING,              // Plik musi istnieć
            FILE_ATTRIBUTE_NORMAL,      // Atrybuty standardowe
            NULL );
  if (hFile == INVALID_HANDLE_VALUE) {    printf("CreateFile error %d.\n", GetLastError());
                                          getchar(); return (1); }

DWORD  dwPtr = SetFilePointer(   // ------ustawienie miejsca odczytu w pliku
                        hFile,          // Uchwyt do pliku wcześniej utworzonego
                        OFFSET,         // Liczba bajtów do przesunięcia
                        NULL,           // Używany przy dużych plikach
                        FILE_BEGIN) ;   // Ustawienie na początek pliku
    if (dwPtr == INVALID_SET_FILE) {  printf("SetFilePointer error %d.\n", GetLastError() );
                                      getchar(); return (2); }

BOOL  bResult = ReadFile(     // ------------------------odczyt z pliku
                        hFile,          // Uchwyt pliku
                        Wy,             // Tablica Wyjsciowa
                        BYTEStoREAD,    // Liczba bajtów do wczytania
                        &readed,        // pointer na liczbe rzeczywiście wczytanych bajtów
                        NULL) ;
     if (!bResult) {   printf("ReadFile error %d.\n", GetLastError() ) ; getchar(); return (3); }

if (readed == BYTEStoREAD) printf("Pelny Odczyt zakonczony pomyslnie\n");
    else printf("?? Wczytano %d ze %d bajtow zalozonych.\n", readed, BYTEStoREAD);
CloseHandle(hFile);
puts(Wy);
// getchar();
return 0;
}
```

```
?? Wczytano 56 ze 150 bajtow zalozonych
89 Tekst tego pliku
zawiera tylko dwa wiersze tekstu.
☺Ĺ|< #
```


Zagadka. Co robi program ***Plik1_a*** ?

Wyjaśnić mechanizmy, które prowadzą do danego wyniku.

```
                // plik danych: Plik1_a.txt   musi istniec na dysku
                // zawartosc pliku danych  Plik1_a.txt :  Kanapa Bandur Rendus Murdek Rumbus
#include <windows.h>
#include <cstdlib>
#include<cstdio>
using namespace std;
int main()                                                // Plik1_a
{
DWORD  readed = 0;
HANDLE  hFile;
char S[5][7],   nameFile[33] = "Plik1_a.txt" ;
int i;
hFile = CreateFile( nameFile, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,      NULL );
    if (hFile == INVALID_HANDLE_VALUE) {  printf("CreateFile error %d.\n", GetLastError()) ; getchar(); return (1); }

BOOL bResult = ReadFile( hFile, S,  100,  &readed, NULL) ;
                          if (!bResult) { printf("ReadFile error %d.\n", GetLastError()) ;  getchar(); return (3); }

for (i=0; i<5; i++) S[i][6] = '\0' ;
for (i=0; i<5; i++) puts(S[i]);
CloseHandle(hFile);
return 0;
}
```

Zadanie 1.1

Napisać program, który zapisze do pliku dyskowego wektor 20-elementowy liczb double.

Zadanie 1.1a

Plik dyskowy zawiera 20 liczb typu double, zapisanych w postaci binarnej.
Napisać program, który wprowadzi do wektora:
- całą zawartość piku ,
- sześć pierwszych liczb,
- kilka środkowych liczb.

Zadanie 1.2

Napisać program, który zapisze do pliku dyskowego macierz M[8][11] liczb double.

Zadanie 1.2a

Plik dyskowy zawiera macierz M[8][11] liczb double, zapisaną w postaci binarnej.
Napisać program, który wprowadzi:
- -zawartości całego pliku do macierzy,
- cztery pierwsze wiersze do macierzy,
- szósty wiersz do wektora.

➔ **W zadaniach Operacje Dyskowe realizować wyłącznie funkcjami systemu operacyjnego.**

ANEKS 1. Obsługa portu szeregowego COM

BOOL **GetCommState(**

HANDLE *hFile*, // handle of communications device The CreateFile function returns this handle.

LPDCB *lpDCB* // address of device-control block structure: **DCB** structure

); If the function succeeds, the return value is nonzero.

BOOL **SetCommState(**

HANDLE *hFile*, // handle of communications device

LPDCB *lpDCB* // address of device-control block structure

); If the function succeeds, the return value is nonzero.

typedef **struct _DCB** { // **dcb** setting for a **serial** communications device.

```
DWORD   DCBlength;          // sizeof(DCB)
DWORD   BaudRate;           // current baud rate
DWORD   fBinary: 1;         // binary mode, no EOF check
DWORD   fParity: 1;         // enable parity checking
DWORD   fOutxCtsFlow:1;     // CTS output flow control
DWORD   fOutxDsrFlow:1;     // DSR output flow control
DWORD   fDtrControl:2;      // DTR flow control type
DWORD   fDsrSensitivity:1;  // DSR sensitivity
DWORD   fTXContinueOnXoff:1;// XOFF continues Tx
DWORD   fOutX: 1;           // XON/XOFF out flow control
DWORD   fInX: 1;            // XON/XOFF in flow control
DWORD   fErrorChar: 1;      // enable error replacement
DWORD   fNull: 1;           // enable null stripping
DWORD   fRtsControl:2;      // RTS flow control
DWORD   fAbortOnError:1;    // abort reads/writes on error
DWORD   fDummy2:17;         // reserved
WORD    wReserved;          // not currently used
WORD    XonLim;             // transmit XON threshold
WORD    XoffLim;            // transmit XOFF threshold
BYTE    ByteSize;           // number of bits/byte, 4-8
BYTE    Parity;             // 0-4=no, odd, even, mark, space
BYTE    StopBits;           // 0,1,2 = 1, 1.5, 2
char    XonChar;            // Tx and Rx XON character
char    XoffChar;           // Tx and Rx XOFF character
char    ErrorChar;          // error replacement character
char    EofChar;            // end of input character
char    EvtChar;            // received event character
WORD    wReserved1;         // reserved; do not use
```

} **DCB**;

DB-9	Nazwa	Funkcja	Opis złącza szeregowego
1	DCO	We	Data Carrier Detect -sygnalizacja wykrycia nośnej
2	RXD	We	Receive Data -odbiór danych
3	TxO	Wy	Transmit Data -nadawanie danych
4	DTR	Wy	Data Terminal Ready -gotowość do nadawania danych
5	GND	masa	System Ground -masa
6	DSR	We	Data Set Ready -gotowość do odbioru danych
7	RTS	Wy	Request to Send - żądanie transmisji
8	CST	We	Clear to Send -kasowanie transmisji
9	RI	We	Ring Indicator -sygnał dzwonienia

Program **CommTest1** ilustruje wykorzystanie ***CreateFile*** do komunikacji z portem szeregowym.

```
#include <windows.h>                                        // port szeregowy COM
#include <stdio.h>
int main(int argc, char *argv[ ])                           // CommTes1
{
DCB dcb;
HANDLE hCom;
BOOL OK;
CHAR *CommPort = "COM1";
hCom = CreateFile(
    CommPort,                           // nazwa pliku / Urządzenia
    GENERIC_READ | GENERIC_WRITE,       // tryb zapisu i odczytu
    0,                                  // zablokowane współdzielenie
    NULL,                               // nie określamy atrybutów bezpieczeństwa
    OPEN_EXISTING,                      // należy użyć OPEN_EXISTING
    0,                                  // nie uzywamy trybu overlapped I/O
    NULL                                // dla urządzeń NULL
    );
 if (hCom == INVALID_HANDLE_VALUE) {    // gdy operacja CreateFile  nie powiodła się
       switch (GetLastError( )) {
                case 2:  printf("Nie wykryto portu %s.\n", CommPort);   break;
                case 3:  printf("Port %s jest zajęty", CommPort);       break;
                default:  printf("CreateFile error %d.\n", GetLastError());
                }
      getchar();  return (1);
      }
OK = GetCommState(hCom, &dcb);      // ----------pobiera konfiguracje portu
 if (!OK) {      printf ("GetCommState error %d.\n", GetLastError());  getchar(); return (2);  }

// ----------------DCB: 57,600 bps, 8 data bits, no parity, 1 stop bit.
        dcb.BaudRate = CBR_57600;
        dcb.ByteSize = 8;
        dcb.Parity = NOPARITY;
        dcb.StopBits = ONESTOPBIT;
 OK = SetCommState(hCom, &dcb);  // -------------ustawianie parametrów portu
 if (!OK)  {  printf("SetCommState error %d.\n", GetLastError());  getchar(); return (3);  }
printf("Pomyslna konfiguracja portu %s.\n", CommPort);
CloseHandle(hCom);
// getchar();
return (0);
}
```