Ćwiczenie

„Operacje na plikach i katalogach”

Tematy ćwiczenia

- tworzenie plików i katalogów,
- otwieranie i zapisywanie,
- ustawianie atrybutów plików.

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 można było archiwizować) pliku ze sprawozdaniem musi zawierać nazwę przedmiotu, numer ćwiczenia i nazwisko studenta, na przykład "PN_...".

Pliki ze sprawozdaniem są przekazywane do archiwum grupy.

a) Tworzenie plików i katalogów

Zadanie

Opracować program, który:

1. Tworzy katalog „DANE”.
2. Tworzy w tym katalogu plik z nazwą „test” z atrybutem „do czytania i zapisu”.
3. Zapisuje do pliku 100 liczb całkowitych z zakresu od -99 do +99 wygenerowanych losowo.
4. Wyświetla liczby po 10 w jednym wierszu.
5. Zamyka plik.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu aplikacji>>

<<zrzut ekranu Eksploratora Windows>>

<<zawartość pliku "test">>

---

b) Operacje na plikach

Zadanie

Opracować program, który:

1. Otworzy z atrybutem „tylko czytanie” plik „test” z katalogu „DANE” stworzony w punkcie a).
2. Tworzy w katalogu „DANE” pliki z nazwami „plik1” i „plik2” z atrybutem „do czytania i zapisu”..
3. Przypisuje z pliku „test” do plików „plik1” i „plik2” dane według reguły ...[1] .
4. Wyświetla liczby z plików „plik1” i „plik2” po 10 w jednym wierszu.
5. Zamyka wszystkie pliki.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu aplikacji>>

<<zawartość plików "plik1" i "plik2">>

[1] z tabeli wariantów

Tabela wariantów

Np.	Plik1	Plik2	Np	Plik1	Plik2
1	Każda liczba parzysta	Każda liczba nieparzysta	**17**	Każda liczba parzysta	Co trzecia nieparzysta
2	Co druga parzysta	Co druga nieparzysta	**18**	Co druga parzysta	Co czwarta nieparzysta
3	Co trzecia parzysta	Co trzecia nieparzysta	**19**	Co trzecia parzysta	Co piąta nieparzysta
4	Co czwarta parzysta	Co czwarta nieparzysta	**20**	Co czwarta parzysta	Co szósta nieparzysta
5	Co piąta parzysta	Co piąta nieparzysta	**21**	Co piąta parzysta	Co siódma nieparzysta
6	Co szósta parzysta	Co szósta nieparzysta	**22**	Co szósta parzysta	Co ósma nieparzysta
7	Co siódma parzysta	Co siódma nieparzysta	**23**	Co siódma parzysta	Każda liczba nieparzysta
8	Co ósma parzysta	Co ósma nieparzysta	**24**	Co ósma parzysta	Co druga nieparzysta
9	Każda liczba parzysta	Co druga nieparzysta	**25**	Każda liczba parzysta	Co czwarta nieparzysta
10	Co druga parzysta	Co trzecia nieparzysta	**26**	Co druga parzysta	Co piąta nieparzysta
11	Co trzecia parzysta	Co czwarta nieparzysta	**27**	Co trzecia parzysta	Co szósta nieparzysta
12	Co czwarta parzysta	Co piąta nieparzysta	**28**	Co czwarta parzysta	Co siódma nieparzysta
13	Co piąta parzysta	Co szósta nieparzysta	**29**	Co piąta parzysta	Co ósma nieparzysta
14	Co szósta parzysta	Co siódma nieparzysta	**30**	Co szósta parzysta	Każda liczba nieparzysta
15	Co siódma parzysta	Co ósma nieparzysta	**31**	Co siódma parzysta	Co druga nieparzysta
16	Co ósma parzysta	Każda liczba nieparzysta	**32**	Co ósma parzysta	Co trzecia nieparzysta

Wskazówki

Tworzenie katalogu

Funkcja API Win32 **GetCurrentDirectoryA** z pliku user32.inc podaje bieżący katalog.
Argumenty funkcji:
nBufferLength – rozmiar bufora,
lpBuffer – adres bufora do rozmieszczenia nazwy.
Funkcja zwraca (przez rejestr EAX) długość ciągu znaków (tj. nazwę katalogu ze ścieżką). Ciąg znaków jest zapisany w kodzie ASCII i jest zakończony zerem. Nazwa bieżącego katalogu ze ścieżką nie zawiera ostatniego znaku „\".

Funkcja API Win32 **CreateDirectoryA** z pliku user32.inc tworzy katalog.
Argumenty funkcji:
lpPathName – adres wierszu z nazwą nowego katalogu,
lpSecurityAttributes – adres struktury SecurityAttributes opisującej prawa dostępu do katalogu (można ustawić na 0, jeśli katalog jest „zwykły").
Funkcja **CreateDirectoryA** zwraca 0 w przypadku błędu. Opis błędu można otrzymać wywołując funkcję GetLastError.

Operacje na wierszach

W celu podłączenia do ciągu nazwy katalogu lub pliku można użyć funkcji:
lstrcpyA do kopiowania wierszu (ciągu znaków zakończonych zerem),
lstrcatA do podłączenia wierszu,
lstrlenA do obliczenia długości wierszu.
Funkcja **lstrcpyA** ma argumenty:
lpString1 – adres bufora – miejsca przeznaczenia,
lpString2 – adres wierszu do kopiowania.
Funkcja **lstrcatA** ma argumenty:
lpString1 – adres bufora – miejsca przeznaczenia,
lpString2 – adres wierszu do podłączenia.
Funkcja **lstrlenA** ma argument:
lpString – adres wierszu, którego długość jest do obliczenia.

Tworzenie lub otwarcie pliku

Funkcja API Win32 **CreateFileA** z pliku user32.inc tworzy lub otwiera plik.
Argumenty funkcji:
lpszName - adres nazwy pliku ze ścieżką,
fdwAccess – tryb dostępu do pliku: GENERIC_READ – do odczytu, GENERIC_WRITE – do zapisu, które można połączyć operatorem „OR”,
fdwShareMode – tryb dostępu do pliku ze strony innych aplikacji (można ustawić na 0),
lpsa – adres struktury SECURITY_ATTRIBUTES z informacjami o zabezpieczeniach (można ustawić na 0),
fdwCreate – tryb otwarcia pliku: CREATE_ALWAYS – kreacja nowego pliku, OPEN_EXISTING – otwarcie istniejącego pliku,
fdwAttrsAndFlags – dodatkowe atrybuty (można ustawić na 0),
hTemplateFile – deskryptor pliku tymczasowego (można ustawić na 0).
Funkcja zwraca (przez rejestr EAX) deskryptor (HANDLE) pliku, który należy stosować w funkcjach plikowych.

Zamknięcie pliku

Do zamknięcia pliku służy funkcja API Win32 **CloseHandle** z pliku user32.inc.
Funkcja **CloseHandle** ma argument:
hObject – deskryptor pliku.

Zapisywanie do pliku

Do zapisywania do pliku służy funkcja API Win32 **WriteFile** opisana w pliku user32.inc.
Funkcja **WriteFile** ma argumenty:
hFile – deskryptor pliku.
lpBuffer – adres bufora z danymi,
nNumberOfBytesToWrite – ilość bajtów do zapisywania,
lpNumberOfBytesWritten – adres zmiennej do przechowywania ilości zapisanych bajtów,
lpOverlapped – adres struktury OVERLAPPED z informacją o nadpisaniu (można ustawić na 0).
Funkcja zwraca (przez rejestr EAX) ilość faktyczne zapisanych bajtów.

Odczyt z pliku

Do odczytu z pliku służy funkcja API Win32 **ReadFile** opisana w pliku user32.inc.
Funkcja **ReadFile** ma argumenty:
hFile – deskryptor pliku.
lpBuffer – adres bufora do przyjmowania danych,
nNumberOfBytesToRead – ilość bajtów do odczytu (rozmiar buforu),
lpNumberOfBytesRead – adres zmiennej do przechowywania ilości odczytanych bajtów,
lpOverlapped – adres struktury OVERLAPPED z informacją o nadpisaniu (można ustawić na 0).
Funkcja zwraca (przez rejestr EAX) ilość faktyczne odczytanych bajtów.

Przemieszczenie w pliku

Do przemieszczenia w pliku służy funkcja API Win32 **SetFilePointer** z biblioteki user32.lib.
Funkcja **SetFilePointer** ma argumenty:
hFile – deskryptor pliku.
lDistanceToMove – odległość (w bajtach) do wartości (2^32 – 2),
lpDistanceToMoveHigh – ten argument musi być równy 0, jeśli rozmiar pliku jest mniejszy niż (2^32 – 2), a jeśli rozmiar pliku jest większy niż (2^32 – 2), to ten argument musi być adresem 32-bitowej zmiennej, która razem z argumentem lDistanceToMove tworzy 64-bitową odległość,
dwMoveMethod – opcja wskazująca na regułę liczenia odległości: FILE_BEGIN – odległość jest liczona od początku pliku, FILE_CURRENT – odległość jest liczona od aktualnej pozycji, FILE_END – odległość jest liczona od końca pliku.
Funkcja zwraca (przez rejestr EAX) pozycję wskaźnika pliku. Jeśli rozmiar pliku jest większy niż (2^32 – 2), to argument **lpDistanceToMoveHigh** wskazuje na 32-bitową zmienną, która razem z zawartością rejestru EAX tworzy 64-bitową pozycję.

Generowanie liczb pseudolosowych

Do generowania liczb pseudolosowych służy funkcja **nrandom** z biblioteki MASM32.
Funkcja **nrandom** ma argument typu DWORD – zakres liczb (liczba całkowita nieujemna).
Funkcja zwraca (przez rejestr EAX) liczbę pseudolosową.
Fragment programu:

```
_DATA SEGMENT

liczbaLosowa      DD        0

_DATA SEGMENT
_TEXT SEGMENT

;losowanie liczby od -99 do 99
   invoke nrandom, 198
   sub eax,99
   mov liczbaLosowa, EAX

_TEXT ENDS
```

Wyświetlenie numeru błędu

```
.DATA
  formErr     DB"%d=%xh",0Dh,0Ah,0
  nErr     DD    (?)
  bufor    DB    128 dup (0)
  rbuf     DD    (?)
  rout     DD    (?)
  hout     DD    (?)
.CODE
;--- jeśli blad ----
  call  GetLastError
  mov   nErr,EAX
  INVOKE  wsprintfA,OFFSET bufor,OFFSET formErr,nErr,nErr
  mov   rbuf, EAX
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  push 0        ;; rezerwa, musi być zero
  push OFFSET rout ;;wskaźnik na faktyczną ilość wyprowadzonych znaków
  push rbuf          ;; ilość znaków
  push OFFSET bufor  ;; wskaźnik na tekst
  push hout          ;; deskryptor buforu konsoli
  call WriteConsoleA ;; wywołanie funkcji WriteConsoleA
```

Program przykładowy

```
;Aplikacja z operacjami nad plikami
.586P
.MODEL flat, STDCALL
;--- stale z pliku windows.inc ---
STD_INPUT_HANDLE              equ -10
STD_OUTPUT_HANDLE             equ -11
GENERIC_READ                  equ 80000000h
GENERIC_WRITE                 equ 40000000h
CREATE_NEW                    equ 1
CREATE_ALWAYS                 equ 2
OPEN_EXISTING                 equ 3
OPEN_ALWAYS                   equ 4
TRUNCATE_EXISTING             equ 5
FILE_FLAG_WRITE_THROUGH       equ 80000000h
```

```
FILE_FLAG_OVERLAPPED              equ 40000000h
FILE_FLAG_NO_BUFFERING            equ 20000000h
FILE_FLAG_RANDOM_ACCESS           equ 10000000h
FILE_FLAG_SEQUENTIAL_SCAN         equ 8000000h
FILE_FLAG_DELETE_ON_CLOSE         equ 4000000h
FILE_FLAG_BACKUP_SEMANTICS        equ 2000000h
FILE_FLAG_POSIX_SEMANTICS         equ 1000000h
FILE_ATTRIBUTE_READONLY           equ 1h
FILE_ATTRIBUTE_HIDDEN             equ 2h
FILE_ATTRIBUTE_SYSTEM             equ 4h
FILE_ATTRIBUTE_DIRECTORY          equ 10h
FILE_ATTRIBUTE_ARCHIVE            equ 20h
FILE_ATTRIBUTE_NORMAL             equ 80h
FILE_ATTRIBUTE_TEMPORARY          equ 100h
FILE_ATTRIBUTE_COMPRESSED         equ 800h
FORMAT_MESSAGE_ALLOCATE_BUFFER    equ 100h
FORMAT_MESSAGE_IGNORE_INSERTS     equ 200h
FORMAT_MESSAGE_FROM_STRING        equ 400h
FORMAT_MESSAGE_FROM_HMODULE       equ 800h
FORMAT_MESSAGE_FROM_SYSTEM        equ 1000h
FORMAT_MESSAGE_ARGUMENT_ARRAY     equ 2000h
FORMAT_MESSAGE_MAX_WIDTH_MASK     equ 0FFh

;--- funkcje API Win32 z pliku  user32.inc ---
CharToOemA PROTO :DWORD,:DWORD
;--- z pliku .\include\kernel32.inc ---
GetStdHandle PROTO :DWORD
ReadConsoleA PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
WriteConsoleA PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
ExitProcess PROTO :DWORD
wsprintfA PROTO C :VARARG     ;; int wsprintf(LPTSTR lpOut,// pointer to buffer for output
                    ;; LPCTSTR lpFmt,// pointer to format-control string
                    ;;    ...  // optional arguments  );
lstrlenA PROTO :DWORD
GetCurrentDirectoryA PROTO :DWORD,:DWORD
;;nBufferLength, lpBuffer; zwraca length
CreateDirectoryA PROTO :DWORD,:DWORD
;;lpPathName, lpSecurityAttributes; zwraca 0 jeśli błąd
lstrcatA PROTO :DWORD,:DWORD
;; lpString1, lpString2; zwraca lpString1
CreateFileA PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD,:DWORD,:DWORD
     ;; LPCTSTR lpszName, DWORD fdwAccess, DWORD fdwShareMode,
  ;; LPSECURITY_ATTRIBUTES lpsa, DWORD fdwCreate,
     ;; DWORD fdwAttrsAndFlags, HANDLE hTemplateFile
lstrcpyA PROTO :DWORD,:DWORD
;;LPTSTR lpString1 - address of buffer, LPCTSTR lpString2 - address of string to copy
CloseHandle PROTO :DWORD
;; BOOL CloseHandle(HANDLE hObject)
WriteFile PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
   ;; 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 lpOverlapped - pointer to structure needed for overlapped I/O);
ReadFile PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD
    ;;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);
CopyFileA PROTO :DWORD,:DWORD,:DWORD
   ;; BOOL CopyFile(LPCTSTR lpExistingFileName - pointer to name of an existing file,
   ;;LPCTSTR lpNewFileName - pointer to filename to copy to,
  ;;BOOL bFailIfExists - flag for operation if file exists);
GetLastError PROTO

;--- z pliku masm32.inc ---
nrandom     PROTO :DWORD
;--- funkcje
ScanInt PROTO C adres:DWORD
;------------
includelib ..\lib\user32.lib
includelib ..\lib\kernel32.lib
includelib ..\lib\masm32.lib
;-------------
_DATA SEGMENT

...

_DATA ENDS
;------------
_TEXT SEGMENT
start:
...

;----- wywołanie funkcji ExitProcess ---------
  INVOKE         ExitProcess,0

ScanInt PROC C adres
;; funkcja ScanInt przekształca ciąg cyfr do liczby, którą jest zwracana przez EAX
;; argument - zakończony zerem wiersz z cyframi
;; rejestry: EBX - adres wiersza, EDX - znak liczby, ESI - indeks cyfry w wierszu, EDI - tymczasowy
;--- początek funkcji
LOCAL number,znacz
;--- odkładanie na stos
  push     EBX
  push     ECX
  push     EDX
  push     ESI
  push     EDI
;--- przygotowywanie cyklu
  INVOKE lstrlenA, adres
  mov      EDI, EAX        ;ilość znaków
  mov      ECX, EAX        ;ilość powtórzeń = ilość znaków
  xor      ESI, ESI        ; wyzerowanie ESI
  xor      EDX, EDX        ; wyzerowanie EDX
  xor      EAX, EAX        ; wyzerowanie EAX
  mov      EBX, adres
  ;------------
  mov      znacz,0
  mov      number,0
;--- cykl -------------------------
pocz:
   cmp   BYTE PTR [EBX+ESI], 0h   ;porównanie z kodem \0
   jne   @F
   jmp   et4
@@:
   cmp   BYTE PTR [EBX+ESI], 0Dh   ;porównanie z kodem CR
```

```
  jne  @F
  jmp  et4
@@:
  cmp  BYTE PTR [EBX+ESI], 0Ah   ;porównanie z kodem LF
  jne  @F
  jmp  et4
@@:
  cmp    BYTE PTR [EBX+ESI], 02Dh  ;porównanie z kodem '-'
  jne    @F
  mov    znacz, 1
  jmp    nast
@@:    cmp    BYTE PTR [EBX+ESI], '0'      ;porównanie z kodem '0'
  jae    @F
  jmp    nast
@@:    cmp    BYTE PTR [EBX+ESI], '9'      ;porównanie z kodem '9'
  jbe    @F
  jmp    nast
;----
@@:    push   EDX    ; do EDX procesor może zapisać wynik mnożenia
  mov    EAX,number
  mov    EDI, 10
  mul    EDI              ;mnożenie EAX * (EDI=10)
  mov    number, EAX   ; tymczasowo z EAX do EDI
  xor    EAX, EAX      ;zerowanie EAX
  mov    AL, BYTE PTR [EBX+ESI]
  sub    AL, '0'  ; korekta: cyfra = kod znaku - kod '0'
  add    number,EAX    ; dodanie cyfry
  pop    EDX
nast:  inc    ESI
    dec  ECX
    jz   @F
    jmp  pocz
;--- wynik
@@:
et4:
  cmp    znacz,1 ;analiza znacznika
  jne    @F
  neg    number
@@:
  mov    EAX,number
;--- zdejmowanie ze stosu
  pop    EDI
  pop    ESI
  pop    EDX
  pop    ECX
  pop    EBX
;--- powrót
  ret
ScanInt  ENDP

_TEXT  ENDS
END start
```