Cheat Sheet for Windows 32 & Windows 64 primitives

StQ

May 26, 2015

1 Program Structure

Insert lines:

```
#define UNICODE
#define _UNICODE
```

before line

```
#include <windows.h>
```

2 Data Type

- Boolean values: BOOL, i.e., a logic value (TRUE or FALSE) on 32 bits
  ```
  typedef int BOOL;
  ```
- Characters:
  - CHAR: A 8-bit Windows ANSI character (sinonym: CCHAR)
    ```
    typedef char CHAR;
    ```
  - TCHAR: mapped (depending on UNICODE definition) on CHAR, i.e., ANCI C char, 8 bit characters or WCHAR, wchar_T, i.e., 16 bit characters, that is::
    ```
    #ifdef UNICODE
    #define TCHAR WCHAR
    #else
    #define TCHAR CHAR
    #endif
    ```
- Strings, are defined with:
  - 8-bit characters "...".
  - 16-bit characters L"...".
  - or generic _T("..."), that is, mapped on 8 or 16-bit strings.
- Unsigned integers
  - DWORD: A 32 bit unsigned integer
    ```
    typedef unsigned long DWORD;
    ```
  - DWORDLONG: A 64 bit unsigned integer
    ```
    typedef unsigned __int64 DWORDLONG;
    ```
  - ULONGLONG: A 64 bit unsigned int
  - also possible: DWORD32, DWORD64, etc.
- Signed integers
  - INT: A 32-bit signed integer
    ```
    typedef int INT;
    ```
  - LONG: A 32 bit signed integer
    ```
    typedef long LONG;
    ```

  - LONGLONG: A 64 bit signed integer
    ```
    typedef __int64 LONGLONG;
    ```
  - LARGE_INTEGER: A 64 bit signed integer. The data has two fields
    ```
    x.LowPart = DWORD
    x.HighPart = LONG
    ```
    in union with
    ```
    x.QuadPart = LONGLONG
    ```
    use the one you need.
  - Other possibile types: INT8, INT16, INT32, INT64, LONG32, LONG64, etc.

- FLoating points: FLOAT, defined as
  ```
  typedef float FLOAT;
  ```

- Pointers
  - PBOOL: A pointer to a BOOL
    ```
    typedef BOOL *PBOOL;
    ```
  - LPBOOL: A pointer to a BOOL
    ```
    typedef BOOL far *LPBOOL;
    ```
  - LPSTR: A Long Pointer To STRing
    ```
    typedef TCHAR *LPSTR;
    ```
    Also available: LPTSTR.
  - LPCSTR: A Long Pointer Constant To STRing
    ```
    typedef const CHAR *LPCSTR;
    ```
    Also available: LPCTSTR.
  - LPWORD: A pointer to a WORD
    ```
    typedef WORD *LPWORD;
    ```
  - LPDWORD: A pointer to a DWORD
    ```
    typedef DWORD *LPDWORD;
    ```
  - PLONG: A pointer to a LONG
    ```
    typedef LONG *PLONG;
    ```
  - LPLONG: A pointer to a LONG
    ```
    typedef long *LPLONG;
    ```
  - LPVOID: A Long Pointer to VOID PVOID (L for long is superfluous)
  - also possible: PINT8, PINT16, PINT32, etc., PLONG32, etc., PLONGLONG, etc.

- Handles
  - HANDLE: generic object (for processes, threads, etc.) handle.
    ```
    typedef PVOID HANDLE;
    ```
  - PHANDLE: a pointer to an handle
    ```
    typedef HANDLE *PHANDLE;
    ```

3 File

3.1 Manipulation Commands (Hart page 46)

Primitives:

```
BOOL DeleteFile (LPCTSTR lpFileName);
BOOL CopyFile (
  LPCTSTR lpExistingFileName,
  LPCTSTR lpNewFileName,
  BOOL fFailIfExists
);
BOOL CreateHardLink (
  LPCTSTR lpFileName,
  LPCTSTR lpExistingFileName,
  BOOL leSecurityAttributes
);
BOOL CreateSymbolicLink (
  LPCTSTR lpSymLinkFileName,
```

```
  LPCTSTR lpTargetFileName,
  DWORD dwFlags
);
BOOL MoveFile (
  LPCTSTR lpExistingFileName,
  LPCTSTR lpNewFileName
);
BOOL MoveFileEx (
  LPCTSTR lpExistingFileName,
  LPCTSTR lpNewFileName,
  DWORD dwFlags
);
```

3.2 Open, close, read, write

Primitives and usage:

```
HANDLE hIn, hOut;
hIn = CreateFile (argv[1], GENERIC_READ, FILE_SHARE_READ, NULL,
  OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
hOut = CreateFile (argv[2], GENERIC_WRITE, 0, NULL,
  CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (hIN == INVALID_HANDLE_VALUE || ...) {
}
while (ReadFile (hIn, buffer, BUF_SIZE, &nIn, NULL) && nIn > 0) {
  WriteFile (hOut, buffer, nIn, &nOut, NULL);
  if (nIn != nOut) {
    fprintf (stderr, "Fatal write error: %x\n", GetLastError ());
    CloseHandle(hIn); CloseHandle(hOut);
    return 4;
  }
}
CloseHandle (hIn);
CloseHandle (hOut);
}
```

3.3 From ASCII to BINARY

Primitives and usage:

```
#ifdef UNICODE
  err = _wfopen_s (&fp, argv[1], _T("r"));
#else
  err = fopen_s (&fp, argv[1], "r");
#endif
if (err != 0) {
  _tprintf (_T("Cannot open output file %s.\n"), argv[1]);
  return 3;
}
_tprintf (_T("Dedug Printing 1 (what I read from ASCCI and write to bin):\n"));
while (_ftscanf (fp, _T("%d%ld%s%s%d"),
  &myse.id, &myse.rn, myse.n, myse.s, &myse.mark) != EOF) {
  WriteFile (hOut, &myse, sizeof (struct mys), &nOut, NULL);
}
```

3.4 Overlapped

Primitives and usage:

```
OVERLAPPED ov = {0, 0, 0, 0, NULL};
LARGE_INTEGER filePos;
filePos.QuadPart = (n-1) * sizeof (struct mys);
#if SETFILEPOINTER_OVERLAPPING
  SetFilePointerEx (h, filePos, NULL, FILE_BEGIN);
  ReadFile (h, &myse, sizeof (struct mys), &n, NULL);
#else
  ov.Offset = filePos.LowPart;
  ov.OffsetHigh = filePos.HighPart;
  ReadFile (h, &myse, sizeof (struct mys), &n, &ov);
#endif
```

3.5 Lock File

Primitives and usage:

```
OVERLAPPED ov = {0, 0, 0, 0, NULL};
fileReserved.QuadPart = 1 *sizeof (files_t);
filePos.QuadPart = (fileDataOperation.id-1) * sizeof (files_t);
ov.Offset = filePos.LowPart;
ov.OffsetHigh = filePos.HighPart;
ov.hEvent = 0;
LockFileEx (hAccount, LOCKFILE_EXCLUSIVE_LOCK, 0,
  fileReserved.LowPart, fileReserved.HighPart, &ov);
ReadFile (hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
UnlockFileEx (hAccount, 0, fileReserved.LowPart,
 fileReserved.HighPart, &ov);
```

4 Directory

4.1 Primitives (Hart page 70)

```
CreateDirectory
RemoveDirectory
SetCurrentDirectory
GetCurrentDirectory
```

4.2 Visit a tree (Hart page 70)

Primitives:

```
HANDLE FindFirstFile (
  LPCTSTR lpFileName,
  LPWIN32_FIND_DATA lpfdd
);
BOOL FindNextFile (
  HANDLE hFindFile,
  LPWIN32_FIND_DATA lpfdd
);
BOOL FindClose (HANDLE hFindFile);
BOOL GetFileTime (
  HANDLE hFile,
  LPFILETIME lpftCreation,
  LPFILETIME lpftLastAccess,
  LPFILETIME lpftLastWrite
);
DWORD GetFileAttribute (LPCTSTR lpFileName);
```

Usage:

```
LPTSTR path,
LPTSTR SourcePathName,
LPTSTR FullDestPathName
CreateDirectory (FullDestPathName, NULL);
SetCurrentDirectory (SourcePathName);
SearchHandle = FindFirstFile (_T("*"), &FindData);
do {
  FType = FileType (&FindData);
  l = _tcslen(FullDestPathName);
  if (FullDestPathName[l-1] == '\\') {
    _stprintf (fullNewPath, _T("%s%s"),
      FullDestPathName, FindData.cFileName);
  } else {
    _stprintf (fullNewPath, _T("%s\\%s"),
      FullDestPathName, FindData.cFileName);
  }
  if (FType == TYPE_FILE) {
    MyCopyFile (FindData, fullNewPath);
  }
  if (FType == TYPE_DIR) {
    TraverseAndCreate (path, FindData.cFileName, fullNewPath);
    SetCurrentDirectory (_T (".."));
  }
} while (FindNextFile (SearchHandle, &FindData));
FindClose (SearchHandle);
```

4.3 Check Entry Type

```
static DWORD
FileType (
  LPWIN32_FIND_DATA pFileData
  )
{
  BOOL IsDir;
  DWORD FType;
  FType = TYPE_FILE;
  IsDir = (pFileData->dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0;
  if (IsDir)
    if (lstrcmp (pFileData->cFileName, _T (".")) == 0
        || lstrcmp (pFileData->cFileName, _T ("..")) == 0)
      FType = TYPE_DOT;
    else FType = TYPE_DIR;
  return FType;
}
```

5 Threads

5.1 Define a Function Thread

```
typedef struct threads {
  ...
} threads_t;
DWORD WINAPI threadFunction (LPVOID);
DWORD WINAPI threadFunction (LPVOID lpParam) {
```

```
  threads_t *data;
  data = (threads_t *) lpParam;
  ...
#if THREAD_CALL
  _endthreadex (0);
#else
  ExitThread (0);
#endif
}
```

5.2 Detach Threads

```
for (i=0; i<argc-2; i++) {
  threadData[i].nameAccount = argv[1];
#if THREAD_CALL
  hThread[i] = (HANDLE) _beginthreadex (NULL, 0, threadFunction,
    &threadData[i], 0, &threadId[i]);
#else
  hThread[i] = CreateThread (NULL, 0,
    (LPTHREAD_START_ROUTINE) threadFunction, &threadData[i],
    0, &threadId[i]);
#endif
  if (hThread[i] == NULL) {
    ExitProcess(0);
  }
}
```

5.3 Wait for Threads

Wait for a single thread:

```
WaitForSingleObject (&hTread, INFINITE);
CloseHandle (hThread);
```

Wait for N threads:

```
// Wait until all threads have terminated.
WaitForMultipleObjects (N, hThread, TRUE, INFINITE);
for (i=0; i<N; i++) {
  CloseHandle (hThread[i]);
}
```

5.4 Suspend a thread

Unclean termination (no deallocation, file closure, etc.) ... Transfer the threads a termination code (set a global variable, i.e., a flag, to a specific value) and instead of managing the current number perform the usual ExitThread instead

```
TerminateThread (hThread[i], 0);
```

6 Synchronization Strategies

6.1 Critical Section

```
CRITICAL_SECTION cs;
InitializeCriticalSection (&cs);
EnterCriticalSection (&cs);
...
LeaveCriticalSection (&cs);
DeleteCriticalSection (&cs);
TryCriticalSection (&cs);
```

6.2 Mutex

More powerfuill than CS. It is possible to time-out (second parameter of wait). Slower than CS.

```
HANDLE mt;
mt = CreateMutex (NULL, FALSE, NULL);
// ... wait ... WaitForSingleObject (mt, INFINITE);
// ... wait ... WaitForMultipleObject (#wait, mtArray, TRUE/FALSE, INFINITE);
// iff TRUE = wait for all; FALSE = wait for jusst one
ReleaseMutex (mt);
```

6.3 Semaphore

```
HANDLE se;
se = CreateSemaphore (NULL, 1, 1, NULL);
// ... wait ... WaitForSingleObject (se, INFINITE);
// If more than one Wait (one ++ on the counter) is necessary
// it is required to pyut a mutex around the two, or more, wait
// (see Hart page 286)
ReleaseSemaphore (se, 1, NULL);
```

	autoReset	manualReset
setEvent	1T now or the first one i.e., 1T, t=[0,infty], reset	all T until explicitly reset i.e., nT, t=[0,infty], to be reset manually
pulseEvent	1T but jut now (not in the future) i.e., 1T, t=[0], reset	all T waiting then reset i.e., nT, t=[0], to be reset manually

7 Event

```
HANDLE CreateEvent (
  LPSECURITY_ATRTRIBUTES lpsa,
  BOOL bManualReset,
  BOOL bInitialState,
  LPTCSTR lpEventName
);
// Signal everybody till a reset
BOOL SetEvent (HANDLE hEvent);
// Stop signalling (after a SetEvent)
BOOL ResetEvent (HANDLE hEvent);
// Signal everybody when issued, then reset signalling
BOOL PulseEvent (HANDLE hEvent);
```

Signalling an event means releasing it to multiple threads.

- manual reset: signal different T, reset by the T
- auto reset: signal 1 T, reset automatically
- event: pulse event, set event

Four different behaviours are possibile:
Usage:

```
HANDLE ev, se;
/* Pulse Event + Initially Not Signalled */
ev = CreateEvent (NULL, TRUE, FALSE, NULL);
se = CreateSemaphore (NULL, 0, 4, NULL);
// INITIALLY Wait to have all Threads Running !!!
Sleep (1000);
/* Wake-up All Threads (they have to be waiting) */
PulseEvent (ev);
```

8 Exception Handling (Hart page 108)

Example:

```
__try {
  monitored code
  (anche le funzioni chiamate sono monitorate)
}
__ except (filter) {
  exception handler
}
```

Where filter is a constant or a function returning a constant value. Available constant values:

- `EXCEPTION_EXECUTE_HANDLER` execute the except code
- `EXCEPTION_CONTINUE_SEARCH` look for a filter which matches at the current identation level
- `EXCEPTION_CONTINUE_EXECUTION` ignore the error and go on with the try code.

The error code can be capture by the `GetExceptionCode (void)` function and dealt with the filter function (see Hart page 108, Figure 4.2): It is possibile to use `RaiseException` to raise exception in an explicit way.
Example:

```
__try {
  monitored code
  (anche le funzioni chiamate sono monitorate)
}
__ finally {
  termination handler
}
```

Like except but with two differences:

- There is no filtering function.
- It is executed even when a `break`, `continue`, `return`, or a `finally` block is executed.

Note that for each `try` there is only one `except` or a single `finally` not both. Obviusly it is possibile to nest the construct.

Usage:

```
__try{
  __try{
    while (
      if (n2 == 0)
        RaiseException (EXCEPTION_FLT_DIVIDE_BY_ZERO, 0, 0, NULL);
      }
    }
    _finally{
      _tprintf(_T("Finally Block Reached (in toDivide Function).\n"));
      CloseHandle (hFile);
      }
    }__except(EXCEPTION_EXECUTE_HANDLER){
     _tprintf(_T("Except Block Reached (in toDivide Function).\n"));
     exception = true;
    }
}
```

9 Process

```
if (!CreateProcess (NULL, commandLine, NULL, NULL,
  TRUE, 0, NULL, NULL, &StartUp, &ProcessInfo))
  _tprintf (_T ("Created Process Server <server.exe>): FAILURE.\n"));
else
  _tprintf (_T ("Created Process Server <server.exe>): SUCCESS.\n"));
```