Introducere în WIN32 API

CURS NR. 1

Windows

Sistem de operare – familie de sisteme de operare

- Gestionează resursele sistemului de calcul și facilitează utilizarea eficientă și ușoară a sistemului
  - Ex. de resurse: memoria, sistemul de fișiere, procesorul, dispozitivele de i/o, etc.

API – Application Programming Interface – suportat de toate versiunile Windows

- Filozofia Windows: are propriile convenții și tehnici de programare
- Oferă suport pentru realizarea obiectivelor sistemului de operare Windows și pune la dispoziție facilitățile SO pe diverse viersiuni de Windows
- Win API este caracterizat de stabilitate și compatibilitate între diferitele versiuni de Windows
  - Nucleul API-ului este proprietar, dar disponibil pe o gamă largă de sisteme de calcul

Win API vs. POSIX și C standard

- Win API nu se conformează standardelor din industrie – nu este compatibil cu standardul POSIX (adoptat de UNIX și Linux)
- Oferă suport limitat pentru standardul C – compatibilitate cu C89, parțial cu C99
- Oferă suport pentru standardele de interoperabilitate esențiale

Win API

Principii Windows

- API-ul Windows diferă semnificativ de alte standarde – ex. API-ul POSIX
  - Necesită un stil de codificare și tehnică aparte
- Caracteristici
  - Reprezentarea celor mai multe resurse sistem ca **obiecte kernel** – identificate printr-un handle
    - Obiectele kernel se gestionează doar prin Win API
  - **Interfață bogată** și flexibilă
    - Multe funcții cu operații similare, funcții care grupează mai multe funcții – pentru comoditate (convenince)
    - Funcții cu număr mare de parametri
  - Gama largă de mecanisme de comunicare și sincronizare
  - **Numele funcțiilor sunt lungi și descriptive** (ex. CreateFile, GetCurrentDirectory,)
  - Numele **tipurilor predefinite** se scriu cu **litere mari**
    - Ex. BOOL (definit ca un obiect de 32 de biți – care stochează o valoare logică)
    - HANDLE (handle pentru un obiect kernel)
    - DWORD (întreg fără semn pe 32 biți)
  - Tipurile predefinite **pointer înglobează operatorul ***
    - LPTSTR (definit ca `TCHAR *`)
    - LPCTSTR (definit ca și `const TCHAR *`)
  - Indentificatorii – mai ales în prototipurile funcțiilor urmează **notația maghiară** (Hungarian notation)
    - Numele include informații despre identificator (semantica, tipul, dacă este referință, dacă este constantă, etc.)
  - Fișiere header: `windows.h`

Tipuri predefinite

Câteva exemple

BOOL	A Boolean variable (should be **TRUE** or **FALSE**). This type is declared in WinDef.h as follows: `typedef int BOOL;`
DWORD	A 32-bit unsigned integer. The range is 0 through 4294967295 decimal. This type is declared in IntSafe.h as follows: typedef unsigned long DWORD;
INT	A 32-bit signed integer. The range is -2147483648 through 2147483647 decimal. This type is declared in WinDef.h as follows: typedef int INT;

INT_PTR	A signed integer type for pointer precision. Use when casting a pointer to an integer to perform pointer arithmetic.
	This type is declared in BaseTsd.h as follows:
	
	`#if defined(_WIN64)`
	`typedef __int64 INT_PTR;`
	`#else`
	`typedef int INT_PTR;`
	`#endif`

LPCSTR	A pointer to a constant null-terminated string of 8-bit Windows (ANSI) characters. For more information, see Character Sets Used By Fonts. This type is declared in WinNT.h as follows: typedef __nullterminated CONST CHAR *LPCSTR;
LPCVOID	A pointer to a constant of any type. This type is declared in WinDef.h as follows: typedef CONST void *LPCVOID;
LPCWSTR	A pointer to a constant null-terminated string of 16-bit Unicode characters. For more information, see Character Sets Used By Fonts. This type is declared in WinNT.h as follows: typedef CONST WCHAR *LPCWSTR;

LPDWORD	A pointer to a DWORD. This type is declared in WinDef.h as follows: typedef DWORD *LPDWORD;
PINT	A pointer to an INT. This type is declared in WinDef.h as follows: typedef int *PINT;
SHORT	A 16-bit integer. The range is –32768 through 32767 decimal. This type is declared in WinNT.h as follows: typedef short SHORT;
SIZE_T	The maximum number of bytes to which a pointer can point. Use for a count that must span the full range of a pointer. This type is declared in BaseTsd.h as follows: typedef ULONG_PTR SIZE_T;
UINT	An unsigned INT. The range is 0 through 4294967295 decimal. This type is declared in WinDef.h as follows: typedef unsigned int UINT;
VOID	Any type. This type is declared in WinNT.h as follows: #define VOID void
WCHAR	A 16-bit Unicode character. For more information, see Character Sets Used By Fonts. This type is declared in WinNT.h as follows: typedef wchar_t WCHAR;
WORD	A 16-bit unsigned integer. The range is 0 through 65535 decimal. This type is declared in WinDef.h as follows: typedef unsigned short WORD;

win32 api și win64 api

Diferența esențială constă în

- dimensiunea variabilei pointer – adrese pe 32 de biți vs. adrese pe 64 de biți
  - Atenție: un întreg și un pointer nu au neaparat aceeași dimensiune
- dimensiunea spațiului de adrese virtual

Executabilele dezvoltate sub win32 pot fi rulate pe sisteme cu 64 de biți – cu limitarea că nu pot beneficia de spațiul de adrese extins specific executabilelor win64

Biblioteca C este disponibilă iar funcțiile care gestionează resursele sistemului sunt implementate în fundal folosind apeluri sistem specifice API-ului Windows

- Când folosim abordarea nativă Windows și când este mai adecvată utilizarea bibliotecii C?
  - **Biblioteca C** când dorim portabilitate mai bună pe alte sisteme decât Windows
    - În cazul programelor simple, care folosesc doar resurse gestionabile doar prin funcțiile oferite de bibliotecă (ex. Fișiere, dar nu procese, etc.) – dar opțiunile sunt limitate
  - **API-ul Windows**
    - În cazul programelor care folosesc resurse gestionabile doar prin API-ul Windows, sau vor să beneficieze de performanța și flexibilitatea abordării native Windows

Noi vom folosi atât biblioteca C cât și extensiile Windows

Sistemul de fișiere Windows

Fișier – pentru stocarea nevolatilă a informațiilor – pe sisteme de memorare secundare (HDD, SSD, etc.)

Realizează gestiunea fișierelor pe dispozitivele de memorare nonvolatile atașate la sistemul de calcul

- Controlează modul de alocare a spațiului, stocarea, regăsirea, controlul accesului, etc.

Oferă suport pentru

- Sistemul de fișiere NTFS (New Technology File System)
  - Suport pentru fișiere de dimensiuni mari (peste 4GB), nume lungi
  - Suport pentru protecție și securitate: controlul accesului prin ACL (Access Control List), criptare, compresie, etc.
- Sistemul de fișiere FAT și FAT32 (File Allocation Table)
  - În FAT32 limita maximă pentru dimensiunea unui fișier este 4GB, numele de fișier de 8 caractere, cu extensia de 3 caractere
  - Nu oferă suport pentru mecanisme de protecție și securitate
- Sistemul de fișiere UDF (Universal Disk Format) numit și Live File System
  - Suport pentru CD-ROM și DVD

Win32 API accesează sistemul de fișiere în mod uniform – cu limitările impuse de SF

Sistemul de fișiere

Structură ierarhică – fiecare partiție este un arbore

- Rădăcina este numele drive-ului
  - A: și B: sunt rezervate dischetelor,
  - C: și D: șamd. sunt de regulă hdd/ssd-uri, DVD-uri și alte dispozitive atașate direct, iar
  - drive-urile de rețea au asociate litere de la finalul alfabetului

- Separatorul dintre directoare și subdirectoare sau fișiere este simbolul \ (backslash)
  - În unele funcții ale API-ului de nivel coborât prametrul cale acceptă și separatorul / (slash)

Acces la fișiere: căi de acces

- Căi absolute
  - Încep cu numele drive-ului
  - Sau încep cu \\ (dublu backslash) indicând o rădăcină globală și este urmată de numele unui server și numele de partajare (share name)
    - Prima parte a căii absolute arată astfel: \\server\sharename

- Căi relative
  - Relative la directorul curent

- . și .. sunt nume de directoare
  - . referă directorul curent
  - .. referă directorul părinte

Sistemul de fișiere

Reguli de numire pentru fișiere și directoare

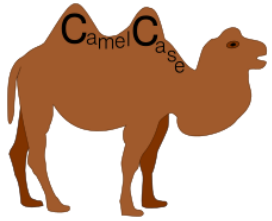
- Numele de directoare și fișiere NU pot conține caractere speciale din intervalul codurilor ASCII 1-31 sau caractere cum ar fi < > : “ | ? * / \
  - aceste caractere au semnificații speciale când sunt folosite în linia de comandă
- Numele poate conține spațiu
  - Numele care conține spațiu se pune între ghilimele - dacă se specifică din linia de comandă
    - Pentru a nu fi parsate ca două fișiere separate
- Numele este case-insensitive dar case-retaining
  - Un fișier cu numele MyFile va fi listat ca atare, dar poate fi accesat și sub nume de genul myfile sau myFILE
- Lungimea numelui: maxim 255 de caractere
- Lungimea unei căi de acces: MAX_PATH caractere (de regulă 260 de caractere)
- Extensia (de regulă între 2-4 caractere) care indică tipul fișierului
  - Un punct separă numele fișierului de extensie (punctul cel mai din dreapta – numele poate conține mai multe puncte)
    - Ex. Sort.exe desemnează un executabil, fis.c este un fișier sursă C

Windows API vs CRT (C Runtime Library)

Resursele unui sistem de operare pot fi accesate doar prin apeluri sistem – API-ul specific SO

Windows API este un set de funcții C prin care sistemul de operare permite accesul la resurse

PSDK – Platform Software Development Kit



- Permite scrierea aplicațiilor Windows
- Include
  - Fișiere header cu declarațiile funcțiilor din Windows API
  - Fișiere lib (pentru linkeditare când funcțiile din API sunt redirectate către DLL-uri)
  - Unelte ajutătoare
- Numele funcțiilor urmează convenția de numire **Camel case**

- Exemplu: pentru crearea unui fișier folosim funcția CreateFile declarată în fișierul header WinBase.h – care necesită biblioteca Kernel32.lib pentru linkeditare

C Runtime Library

- Set de funții C și fișiere header care implementează sarcini comune, de exemplu gestiunea stringurilor, operații de intrare/ieșire, etc.
- Implementate peste nivelul de API – folosesc apelurile sistem din API pentru a accesa resursele sistemului
- Urmează într-o măsură mai mare sau mai mică standardul C
  - Numele funcțiilor se scriu cu litere mici
  - Numele funcțiilor extensie încep cu caracterul _ (underscore)
    - Exemplu: _mkdir

Codificarea caracterelor

Seturi de caractere

Single Byte Character Set

- Setul de caractere **ASCII** cu 1 octet/caracter nu este suficient de încăpător (mai ales pentru unele limbi non-europene)
- Multibyte Character Set: unele caractere ocupă 1 octet, altele doi octeți - problematic
- Soluție: setul de caractere **Unicode** – set multilingual – include codificările UTF16 - cu reprezentarea caracterelor pe 2 octeți (16 biți), UTF32, UTF8, etc.

- Suport pentru Unicode în standardul C
  - Începând cu **C99** standardul oferă suport pentru caractere extinse (wide character) – Unicode
    - Tipul wchar_t din fișierul header wchar.h
  - Începând cu **C11** standardul furnizează caractere cu număr specificat de biți în reprezentare
    - Tipul char16_t și char32_t din fișierul header uchar.h

- Win API oferă suport pentru setul de caractere Unicode
  - Fiecare caracter este codificat folosind UTF-16 (Unicode Transformation Format) ca 2 octeți (16 biți)
  - Cele mai multe funcții din Win32 API au două versiuni pentru a permite preluarea argumentelor ca șiruri ANSI (sufix A) și ca șiruri Unicode (sufix W)
    - Exemplu: CreateFile este o macrodefiniție care se expandează în funcția CreateFileA sau CreateFileW în funcție de parametrul primit

W de la “wide”

Table 2-1: Unicode Character Sets and Alphabets

Open table as spreadsheet

16-Bit Code	Characters	16-Bit Code	Alphabet/Scripts
0000-007F	ASCII	0300-036F	Generic diacritical marks
0080-00FF	Latin1 characters	0400-04FF	Cyrillic
0100-017F	European Latin	0530-058F	Armenian
0180-01FF	Extended Latin	0590-05FF	Hebrew
0250-02AF	Standard phonetic	0600-06FF	Arabic
02B0-02FF	Modified letters	0900-097F	Devanagari

Codificarea caracterelor

Windows API oferă suport pentru

- Caractere pe 8 biți (numite ASCII sau ANSI)
  - Tipul `char` sau `CHAR`
    - Constanta șir de caractere: `"This string uses 8-bit characters"`
- Caractere extinse pe 16 biți (Unicode folosind codificarea UTF-16)
  - Tipul `wchar_t` sau `WCHAR`
    - Constanta șir de caractere: `L"This string uses 16-bit characters"`

Tipul de caracter text generic

- Pentru scrierea aplicațiilor care pot lucra cu ambele seturi de caractere
  - Tipul `TCHAR`
    - Constanta șir de caractere:

```
_T("This string uses generic text characters")
_TEXT("This string uses generic text characters")
TEXT("This string uses generic text characters")
```

Windows API folosește caractere Unicode pe 16 biți

- Numele de fișiere și directoare sunt reprezentate intern în Unicode

Seturi de caractere

Abordarea **ANSI** vs. **Unicode**

Caracterul L din fața șirului indică compilatorului să folosească setul de caractere Unicode

```
// An 8-bit character
char c = 'A';
// An array of 9 8-bit characters
// the 9th character is an 8-bit null character
char szBuffer[] = "A String";
```

```
// A 16-bit character
wchar_t c = L'A';
// An array of 9 16-bit characters
// the 9th character is an 16-bit null character
wchar_t szBuffer[] = L"A String";
```

Tablou de 9 întregi de tip `wchar_t`

```
{L'A', L' ', L'S', L't', L'r', L'i', L'n', L'g', 0}
```

Pentru a oferi o abordare uniformă s-a definit un tip generic pentru un caracter de text: **TCHAR**

- Și alte macrodefiniții relevante definite în fișierul header WinNT.h

```
typedef char CHAR; // An 8-bit character
typedef wchar_t WCHAR; // A 16-bit character

// Pointer to 8-bit character(s)
typedef CHAR *PCHAR;
typedef CHAR *PSTR;
typedef CONST CHAR *PCSTR

// Pointer to 16-bit character(s)
typedef WCHAR *PWCHAR;
typedef WCHAR *PWSTR;
typedef CONST WCHAR *PCWSTR;
```

```
#ifdef UNICODE
   typedef WCHAR TCHAR, *PTCHAR, *PTSTR;
   typedef CONST WCHAR *PCTSTR;

   #define __TEXT(quote) quote // r_winnt
   #define __TEXT(quote) L##quote
#else
   typedef CHAR TCHAR, *PTCHAR, *PTSTR;
   typedef CONST CHAR *PCTSTR;

   #define __TEXT(quote) quote
#endif

#define TEXT(quote) __TEXT(quote)
```

Tipul de caractere text generic: TCHAR

TCHAR se expandează la `char` sau `wchar_t` în funcție de definirea simbolului UNICODE la compilarea codului

- Deci nu se folosește explicit tipul `char` sau `wchar_t`

la crearea unui nou proiect cu VisualStudiu se definește implicit constanta simbolică UNICODE

```
char cResponse; // 'Y' or 'N'
char sUsername[64];
// str* functions
```

```
wchar_t cResponse; // 'Y' or 'N'
wchar_t sUsername[64];
// wcs* functions
```

Folosim tipul generic

```
#include<TCHAR.H> // Implicit or explicit include

TCHAR cResponse; // 'Y' or 'N'
TCHAR sUsername[64];
// _tcs* functions
```

Codul devine independent de modul de codificare a caracterelor

Același principiu se aplică și la nivelul funcțiilor de lucru cu șirurile de caractere

- Pentru independența de codificarea folosită, se recomandă utilizarea macrourilor specifice

Funcții pentru caractere ANSI
Exemple: strcpy, strcat, strlen, și altele pe șablonul **str***

```
size_t strlen(const char*);
```

Funcții pentru caractere extinse (Unicode)
Exemple: wcscpy, wcscat, wcslen, și altele pe șablonul **wcs***

```
size_t wcslen(const wchar_t* );
```

Funcția generică

```
#ifdef _UNICODE
#define _tcslen wcslen
#else
#define _tcslen strlen
#endif
```

Macrodefiniții generice
Exemple: _tcscpy, _tcscat, _tcslen, și altele pe șablonul **_tcs***

```
size_t _tcslen(const TCHAR* );
```

Tipuri generice și funcții generice

Folosim tipuri generice și funcții generice

- Pentru uniformitate și compatibilitate cu versiuni mai vechi de Windows
- În loc de versiunile de funcții cu sufixul A sau W din Win32API, apelăm de fapt o macrodefiniție care se expandează la funcția potrivită la compilare

```
// If UNICODE defined, a 16-bit character; else an 8-bit character
TCHAR c = TEXT('A');
// If UNICODE defined, an array of 16-bit characters; else 8-bit characters
TCHAR szBuffer[100] = TEXT("A String");
```

```
/* GetCurrentDirectory example */
#include <stdio.h>
#include <windows.h>

#define MAX_PATH 50


int main() {
   // Generic code

   PCTSTR psz = TEXT("Hello World!");
   TCHAR szDir[MAX_PATH] = { 0 };
   GetCurrentDirectory(MAX_PATH, szDir);
```

PCTSTR se citește: pointer to constant TCHAR string

```
// What actually happens if UNICODE symbol is NOT defined for a build
const char* psz = "Hello World!";
char szDir[MAX_PATH] = { 0 };
GetCurrentDirectoryA(MAX_PATH, szDir);

// GetCurrentDirectoryA is a wrapper. It does the following:
// 1. Allocates temporary wchar_t buffer of given size.
// 2. Converts the char string into a wchar_t string.
// 3. Calls real worker: GetCurrentDirectoryW.
```

```
// What actually happens if UNICODE symbol is defined for a build
//
const wchar_t* psz = L"Hello World!";
wchar_t szDir[MAX_PATH] = { 0 };
GetCurrentDirectoryW(MAX_PATH, szDir);
// direct call to real worker; no wrappers in the middle
```

Scrierea aplicațiilor Windows generice

Definirea caracterelor și șirurilor de caractere folosind tipuri generice

- Tipul `TCHAR`
- Tipul pointer la un șir generic `LPTSTR`
- Tipul pointer la un șir generic constant `LPCTSTR`

Definirea variabilelor de preprocesare UNICODE și _UNICODE

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

Controlează definițiile funcțiilor Windows

Controlează biblioteca C

Se definesc înainte de `windows.h`

- Dacă sunt definite TCHAR se expandează la `wchar_t`, dacă nu sunt definite se expandează la `char`

Se include după `windows.h`

Folosirea colecției de funcții C generice pentru gestiunea șirurilor de caractere (din `tchar.h`)

- Exemple: _fgettc, _itot (în loc de itoa), _stprintf (în loc de sprintf), _tcscpy (în loc de strcpy), _ttoi, _ttoupper, _ftprintf, etc.

_ indică faptul că aceste funcții sunt extensii furnizate de Microsoft C

Modul de funcționare depinde de existența definiției pentru _UNICODE

- Observație: există și colecții de funcții mai sigure pentru gestiunea stringurilor (Secure CTR și StrSafe) care vor fi prezentate în curând

Funcția principală generică

- Macrodefiniție definită în tchar.h
  - Se expandează la `main` sau `wmain` în funcție de _UNICODE

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

int _tmain(int argc, LPTSTR argv[]) {
    ...
}
```

Aplicații Windows

Windows suportă două tipuri de aplicații

- Bazate pe interfață utilizator de tip consolă – CUI (Console User Interface)
  - Interfață text
- Bazate pe interfață utilizator grafică - GUI (Graphical User Interface)
  - Ferestre, meniuri, căsuțe de dialog, etc.

Noi ne vom axa pe aplicații în consolă

În Microsoft VS la crearea unui proiect, mediul integrat setează linkeditorul pentru a încadra în executabil subsistemul adecvat

- /SUBSYSTEM:CONSOLE pentru aplicații CUI
- /SUBSYSTEM:WINDOWS pentru aplicații GUI

Funcția principală a aplicației are forma următoare

- pentru aplicații CUI

```
int _tmain( int argc, TCHAR *argv[], TCHAR *envp[]);
```

Putem renunța la acest parametru
– dacă nu ne interesează variabilele de mediu

_tmain este
main – pentru stringuri ANSI
wmain – pentru stringuri Unicode

- Pentru aplicații GUI

```
int WINAPI _tWinMain( HINSTANCE hInstanceExe, HINSTANCE, PTSTR pszCmdLine, int nCmdShow);
```

_tWinMain este
WinMain – pentru stringuri ANSI
wWinMain – pentru stringuri Unicode

Maparea stringurilor în CRT

- Macrourile generice pentru caractere sunt definite în fișierul header **tchar.h**
- Maparea generică este o extensie specifică Microsoft. Extensiile față de standard încep cu caracterul _ (underscore)
- Față de abordarea Win32 API (macrodefinițiile din WinNT.h), C Runtime (prin tchar.h) folosește:
  - Macrodefiniția _T(), _TEXT() sau TEXT() pentru specificarea unei constante șir de caractere
  - Trei moduri de codificare pentru caractere text (seturi de caractere)
    - SBCS – Single Byte Character Set – tipul char clasic – abordarea ANSI
    - _MBCS – Multi Byte Character Set – un caracter reprezentat pe unul sau doi octeți
    - _UNICODE – Setul de caractere Unicode – tipul wchar_t – fiecare caracter pe doi octeți – abordarea implicită in MSVS începând cu v.2005

The Generic Text Mapping in CRT

Generic-text data type or name	SBCS (_UNICODE, _MBCS not defined)	_MBCS defined	_UNICODE defined
_TCHAR	char	char	wchar_t
_T("Hello, World!") or _TEXT("Hello")	"Hello, World!"	"Hello, World!"	L"Hello, World!"
Function name prefix and example: _tcs _tcscat, _tcsicmp	str, _str strcat, _stricmp	_mbs _mbscat, _mbsicmp	wcs, _wcs wcscat, _wcsicmp

- Dacă o funcție așteaptă un string extins, dar se furnizează un șir simplu → conversie
  - Analogic și pentru tipul returnat!
- Recomandare: folosim setul Unicode pentru caractere și stringuri

conversia este costisitoare în timp și spațiu

Maparea stringurilor în CRT

Folosind maparea generică a stringurilor putem dezvolta aplicații atât ANSI, MBCS cât și Unicode pe baza aceluiași cod sursă

Exemplu

```
#include <stdio.h>
#include <TCHAR.h>
#include <time.h>
#include <string.h>

int main() {
   struct tm   *newTime;
   time_t      szClock;

   // Get time in seconds
   time(&szClock);

   // Convert time to struct tm form
   newTime = localtime(&szClock);

   // Generic code; names are not standard,
   // hence the leading underscore.
   _TCHAR message[128] = _T("The time is: ");
   _TCHAR* now = _tasctime(newTime);
   _tcscat(message, now);
   _putts(message);

   // ……
```

```
// What happens if no symbol is defined at all (SBCS).
//
char message[128] = "The time is: ";
char* now = asctime(newTime);
strcat(message, now);
puts(message);
```

```
// What happens if _MBCS symbol is defined (Multi-byte Character Set);
// non-standard names are with the leading underscore.
//
char message[128] = "The time is: ";
char* now = asctime(newTime);
_mbscat(message, now);
puts(message);
```

```
// What happens if _UNICODE symbol is defined (Unicode Character Set);
// non-standard names are with the leading underscore.
//
wchar_t message[128] = L"The time is: ";
wchar_t* now = _wasctime(newTime);
wcscat(message, now);
_putws(message);
```

Erori posibile

Exemplu de cod în care dorim să folosim tipul generic TCHAR – potrivit atât pentru stringuri ANSI cât și Unicode

- Care sunt neregulile?

```
TCHAR s[] = _T("Abcde");
size_t len;
len = strlen(s);
```

Compilare cu ANSI –> UNICODE nu este definit pt build

TCHAR este char
Funcția strlen poate primi parametru șir de caractere ANSI

Compilare cu UNICODE definit pt build

TCHAR este wchar_t
Funcția strlen nu acceptă parametru de tip TCHAR []

'size_t strlen(const char *)': cannot convert argument 1 from 'TCHAR []' to 'const char *'

Initializare:

```
TCHAR s[] = (TCHAR *)"Abcde";
```

```
TCHAR s[] = _T("Abcde");
```

```
TCHAR s[] = _TEXT("Abcde");
```

Apelul funcției:

```
len = wcslen(s);
```

Nu se potrivește pentru șiruri ASCII

Conversie la Unicode (cu mbstowcs sau funcții mai sigure din StrSafe) și apoi wcslen

```
len = _tcslen(s);
```

Gestiunea stringurilor

- În C standard nu există tipul string – șirul de caractere este un tablou de caractere cu '\0' la final

Vom prezenta 4 categorii de funcții pentru lucrul cu stringuri

- **Funcții din C standard – incluse în CRT**
  - Funcțiile standard apelează în fundal funcțiile win api, deci au un overhead asociat
  - Furnizează portabilitate dau au limitări - și pot cauza riscuri de securitate
  - Nu returnează un cod de eroare dacă eșuează

Nu este recomandată

Exemple:
`strcpy, wcscpy, _mbscpy, sprintf, swprintf, etc.`

- **Funcții securizate din CRT**
  - Nu sunt din standardul C – apar ca și extensii specifice Windows – au sufixul `_s`
  - Validează parametri, are parametru dimensiunea bufferului, asigură terminare cu caracterul nul
  - Includ cod de eroare – simplist

Exemple:
`strcpy_s, wcscpy_s, _mbscpy_s, sprintf_s, swprintf_s, etc.`

Nu este recomandată

- **Funcții utilizator și kernel din win32 api**
  - Funcții specifice Windows – au mai puțin overhead decât la CRT – au prefixul `l`

Exemple:
`lstrcpy, lstrlen, lstrcat, etc.`

- **Funcții securizate din StrSafe**
  - Operare securizată:
  - Validează parametri, are parametru dimensiunea bufferului, asigură terminare cu caracterul nul
  - Includ cod de eroare – permite gestiunea adecvată a erorilor
  - Perechi de funcții: cu Cch (dimensiune în număr de caractere) și Cb (dimensiune în număr de octeți)

Exemple:
`StringCchLength, StringCbLength StringCchCopy, StringCbCopy, etc.`

Gestiunea caracterelor și a stringurilor

Corespondențele dintre funcțiile CRT și apelurile Win32 API – pentru stringuri

String Manipulation

CRT	Win32 API
strcat, wcscat	lstrcat
strchr, wcschr	none
strcmp, wcscmp	lstrcmp
strcpy, wcscpy	lstrcpy
strcspn, wcscspn	none
_strdup, _wcsdup	none
strerror	FormatMessage
_strerror	FormatMessage
_stricmp, _wcsicmp	lstrcmpi
strlen, wcslen	lstrlen
_strlwr, _wcslwr	CharLower, CharLowerBuffer
strncat, wcsncat	none
strncmp, wcsncmp	none
strncpy, wcsncpy	none
_strnicmp, _wcsnicmp	none
_strnset, _wcsnset	FillMemory, ZeroMemory
strpbrk, wcspbrk	none
strrchr, wcsrchr	none
_strrev, _wcsrev	none
_strset, _wcsset	FillMemory, ZeroMemory
strspn, wcsspn	none
strstr, wcsstr	none
strtok, wcstok	none
_strupr, _wcsupr	CharUpper, CharUpperBuffer

Fără secured CRT

Character Classification

CRT	Win32 API
isalnum	IsCharAlphaNumeric
isalpha	IsCharAlpha, GetStringTypeW (Unicode)
__isascii	none
iscntrl	none, GetStringTypeW (Unicode)
__iscsym	none
__iscsymf	none
isdigit	none, GetStringTypeW (Unicode)
isgraph	none
islower	IsCharLower, GetStringTypeW (Unicode)
isprint	none
ispunct	none, GetStringTypeW (Unicode)
isspace	none, GetStringTypeW (Unicode)
isupper	IsCharUpper, GetStringTypeW (Unicode)
isxdigit	none, GetStringTypeW (Unicode)
__toascii	none
tolower	CharLower
_tolower	none
toupper	CharUpper
_toupper	none

Gestiunea caracterelor și a stringurilor

Win API furnizează funcții separate pentru setul ASCII și Unicode

- Sufix A pentru ASCII
- Sufix W pentru Unicode

- Ex. StringCbCatEx este disponibil în variantele: StringCbCatEx**A** (pentru ASCII), StringCbCatEx**W** (pentru Unicode)

StringCbCatEx este o funcție generică – va apela varianta cu sufixul A pentru șiruri din caractere pe un octet și varianta cu sufixul W pentru caractere extinse

Pentru conversiea șirurilor de caractere dintre ASCII și Unicode se folosesc funcțiile

- MultiByteToWideChar și WideCharToMultiByte

Secure CTR și StrSafe

Funcțiile care **modifică șirul** de caractere ascund potențiale pericole din cauza gestiunii neadecvate a bufferului – bufferul nu este suficient de mare

- Apare problema coruperii memoriei - buffer overflow

Banned API – lista cu funcțiile “periculoase”

- Exemple: strcpy, strncpy, lstrcpy, gets, strcat, lstrcat, etc.

Problema: `wcscpy` nu primește ca și argument dimensiunea maximă a bufferului (în mod similar cu alte funcții de gestiune a șirurilor)

```
// The following puts 4 characters in a
// 3-character buffer, resulting in memory corruption
WCHAR szBuffer[3] = L"";
wcscpy(szBuffer, L"abc"); // The terminating 0 is a character too!
```

Nici funcțiile cu ‘n’ nu sunt sigure!

- Deși permit specificarea numărului maxim de caractere de scris în bufferul destinație DAR NU garantează terminarea stringurilor cu caracterul nul !!!
  - Trebuie explicit adăugat caracterul nul la final

Explicația: funcția strncpy a fost proiectată să permită inserarea unui șir în interiorul unui alt string – deci nu cu scopul de a securiza gestiunea șirurilor

```
wchar_t buffer[5];
wcsncpy_s(buffer, L"Thisisalongstring", sizeof(buffer));
buffer[sizeof(buffer) - 1] = 0;
```

Soluție: **Secured CRT** sau **StrSafe** – gestiunea securizată a șirurilor de caractere

Secured CRT – pentru stringuri

C Runtime Library furnizează o serie de **funcții mai sigure** pentru gestiunea șirurilor de caractere

- Cele mai multe funcții de gestiune a șirurilor din CRT au variante mai sigure
  - La numele funcției se sufixează "_s"
    - Exemple: strcpy_s, wcslen_s, etc.

```
errno_t strcat_s( char *strDestination, size_t numberOfElements, const char *strSource );
errno_t _tcscpy_s(PTSTR strDestination, size_t numberOfCharacters, PCTSTR strSource);
```

- Funcțiile mai sigure necesită ca parametru lungimea maximă a șirului destinație
  - Macroul _countof (decalrat în stdlib.h) returnează numărul de elemente alocate pentru un tablou de caractere

- Funcțiile mai sigure validează argumentele primite
  - Testează ca pointerii să nu fie NULL, întregii să fie în intervalul permis și bufferul să fie suficient de mare pentru a reține șirul rezultat
  - În caz de eșec setează `errno` la valoarea codului de eroare specific și returnează o valoare de tipul `errno_t` care arată succesul sau eroarea apărută

- **Problema**: funcțiile de fapt nu revin în caz de eșec
  - În configurația debug – afișează o eroare și se termină
  - În configurația `release` – aplicația se termină brusc

Pentru mai mult control folosim StrSafe

```
/* _tcscpy_s example */
#include <stdio.h>
#include <stdlib.h>
#include <TCHAR.h>

int main() {
   TCHAR szBefore[5] = _TEXT("BBBB");
   TCHAR szBuffer[10] = _TEXT("---------");
   TCHAR szAfter[5] = _TEXT("AAAA");

   errno_t result = _tcscpy_s(szBuffer, _countof(szBuffer), _TEXT("0123456789"));
   /* …… */
   return 0;
}
```

10 caractere 11 caractere

Caracterul nul nu încape în szBuffer

Nu efectuează trunchiere!

Se returnează ERANGE și
în szBuffer primul caracter este setat la caracterul nul,
restul caracterelor se setează la o valoare de umplere

Se afișează un mesaj de eroare și programul se termină

StrSafe

StrSafe – gestiunea securizată a șirurilor de caractere

- Impune specificarea unei dimensiuni maxime pentru bufferul care stochează șirul destinație – pentru a evita scrierea după sfârșitul șirului
- Garantează existența terminatorului de șir – caracterul nul – chiar și la trunchiere
- Facilitează testarea succesului – toate funțiile returnează o valoare HRESULT – cu valoare S_OK în caz de succes
- Funcțiile fac diferențiere între număr de caractere (“cch”) și număr de octeți (“cb”) (pot trata atât șiruri Unicode cât și ASCII)
- Efectuează trunchiere - spre deosebire de Secured CRT (semnalează cazul prin valoarea returnată)
- Furnizează unele funcționalități extinse – funcții în versiunea “Ex”

Dimensiunea maximă pentru string este STRSAFE_MAX_CCH caractere: 2,147,483,647 (ANSI sau Unicode)

Includerea fișierului header StrSafe.h – după toate celelalte include-uri

- Implicit se inlude și string.h, dar funcțiile înlocuite de strsafe sunt scoase din uz – nu mai sunt disponibile
  - Pentru a putea folosi funcțiile înlocuite, **înainte** de includerea fișierului header strsafe.h se definește
    `#define STRSAFE_NO_DEPRECATE`
  - Pentru a putea folosi doar funcțiile de contorizare a caracterelor (“cch”), **înainte** de includerea fișierului header strsafe.h se definește
    `#define STRSAFE_NO_CB_FUNCTIONS`
  - Pentru a putea folosi doar funcțiile de contorizare a octeților (“cb”), **înainte** de includerea fișierului header strsafe.h se definește
    `#define STRSAFE_NO_CCH_FUNCTIONS`

Atenție: nu putem defini ambele

StrSafe

- Exemple de funcții – C run-time (CRT) – Windows – Secured CRT - StrSafe

CRT String Function	Windows String Function	Secured CRT String Functions	StrSafe Function
strcat, _mbscat, wcscat (_tcscat)	lstrcat	strcat_s, _mbscat_s, wcscat_s (_tcscat_s)	StringCchCat StringCchCatEx StringCbCat StringCbCatEx
strcmp, _mbscmp, wcscmp (_tcscat)	lstrcmp	(no equivalent function)	(no equivalent function)
strcpy, mbscpy, wcscpy (_tcscpy)	lstrcpy	strcpy_s, mbscpy_s, wcscpy_s (_tcscpy_s)	StringCchCopy StringCchCopyEx StringCbCopy StringCbCopyEx
strlen, _mbslen, wcslen (_tcscat)	lstrlen	strlen_s, _mbslen_s, wcslen_s (_tcscat_s)	StringCchLength StringCbLength

- Valori returnate de funcțiile StrSafe

HRESULT Value	Description
S_OK	Success. The destination buffer contains the source string and is terminated by '\0'.
STRSAFE_E_INVALID_PARAMETER	Failure. The NULL value has been passed as a parameter.
STRSAFE_E_INSUFFICIENT_BUFFER	Failure. The given destination buffer was too small to contain the entire source string.

StrSafe

Prezentare comparativă: concatenarea a două stringuri

+ funcția generică: _tcscpy

CRT

```
char *strcat( char *strDestination, const char *strSource );
wchar_t *wcscat( wchar_t *strDestination, const wchar_t *strSource );
```

win32

```
LPTSTR lstrcat( LPTSTR lpString1, LPTSTR lpString2 );
```

Nu impun o limită maximă pentru dimensiunea bufferului destinație

Secured CRT

```
errno_t strcat_s( char *strDestination, size_t numberOfElements, const char *strSource );
errno_t wcscat_s( wchar_t *strDestination, size_t numberOfElements, const wchar_t *strSource );
```

+ funcția generică: _tcscpy_s

Terminare automată în caz de eroare

Dimensiune în număr de caractere – pt tablou alocat static, poate fi obținut cu `_countof` (din stdlib.h)

Efectuează validarea intrărilor

StrSafe

```
HRESULT StringCchCat( LPTSTR  pszDest, size_t  cchDest, LPCTSTR pszSrc );
HRESULT StringCbCat( LPTSTR  pszDest, size_t  cbDest, LPCTSTR pszSrc );
```

Se garantează că șirul se termină cu caracterul nul. Trunchiere dacă se depășesc limitele

Verisuni (mai) sigure

StrSafe

Exemple: concatenare și copiere

Secured CRT

```c
/* wcscat_s example */
#include <stdio.h>
#include <stdlib.h> // for _countof
#include <wchar.h>

int main() {
   wchar_t safe_copy_str1[] = L"Hello world!";
   wchar_t safe_copy_str2[100] = L"He said: ";

   int r = wcscat_s(safe_copy_str2, _countof(safe_copy_str2), safe_copy_str1);
   if (r != 0) {
      wprintf_s(L"wcscat_s() failed %ld", r);
   }
   wprintf(L"After copy string = %s\n", safe_copy_str2);

   return 0;
}
```

StringCchCat și StringCchCopy sunt funcții generice – vor apela varianta cu sufixul A pentru șiruri din caractere pe un octet și varianta cu sufixul W pentru caractere extinse

Funcțiile extinse de ex. StringCchCopyEx oferă mai multe funcționalități. Vezi exemplu aici

StrSafe

```c
/*StringCchCopy example */
#include <stdio.h>
#include <stdlib.h> // for _countof
#include <wchar.h>
#include <strsafe.h>


int main() {
   wchar_t wsString[128];
   HRESULT Res;

   Res = StringCchCopy(wsString, _countof(wsString), L"Hello ");
   if (Res != S_OK) {
      printf("StringCchCopy Failed: %S\n", wsString);
      exit(-1);
   }

   Res = StringCchCat(wsString, sizeof(wsString), L" World!");
   if (Res != S_OK) {
      printf("StringCchCat Failed: %s\n", wsString);
      exit(-1);
   }

   wprintf_s(L"%s\n", wsString);
   return 0;
}
```

StrSafe

Prezentare comparativă: preluarea șirului de la tastatură

CRT

```
char * gets( char * str );
```

Nu validează dimensiunea șirului

Secured CRT

```
char *gets_s( char *buffer, size_t sizeInCharacters );

wchar_t *_getws_s( wchar_t *buffer, size_t sizeInCharacters );
```

Terminare automată în caz de eroare

Dimensiune în număr de caractere

Verisuni (mai) sigure

StrSafe

```
HRESULT StringCchGets( LPTSTR pszDest, size_t cchDest );

HRESULT StringCbGets( LPTSTR pszDest, size_t cbDest );
```

Se garantează că șirul se termină cu caracterul nul. Trunchiere dacă se depășesc limitele

StrSafe

Exemple: preluare șir

Secured CRT

```c
/* _getws_s example */
#include <stdio.h>
#include <stdlib.h> // for abort
#include <wchar.h>


#define MAX_BUF 10

int main() {
   wchar_t safe_getline[MAX_BUF];

   if (_getws_s(safe_getline, MAX_BUF) == NULL) {
      wprintf_s(L"invalid input.\n");
      abort();
   }

   wprintf_s(L"%s\n", safe_getline);
   return 0;
}
```

Ce se întâmplă dacă introducem un șir mai mare de 9 caractere?

StringCchGets este o funcție generică

StrSafe

```c
/* _getws_s example */
#include <stdio.h>
#include <stdlib.h> // for abort and _countof
#include <wchar.h>
#include <strsafe.h>


int main() {
   wchar_t wsString[10];
   HRESULT Res;

   Res = StringCchGets(wsString, _countof(wsString));

   if (Res != S_OK) {
      wprintf_s(L"StringCchGets Failed: %S\n", wsString);
      abort();
   }

   wprintf_s(L"%s \n", wsString);

   return 0;
}
```

```c
Res = StringCbGets(wsString, sizeof(wsString));
```

Atenție sizeof nu este o soluție bună dacă wsString este un pointer - și nu un tablou alocat static !!!

StrSafe

Dimensiunea șirului de caractere

CRT

```
size_t strlen ( const char * str );

size_t wcslen( const wchar_t *str );
```

Nu specifică dimnesiunea maximă

Secured CRT

```
size_t strnlen( const char *str, size_t numberOfElements );

size_t strnlen_s( const char *str, size_t numberOfElements );

size_t wcsnlen( const wchar_t *str, size_t numberOfElements );

size_t wcsnlen_s( const wchar_t *str, size_t numberOfElements );
```

Dacă șirul e mai lung returnează al doilea parametru

Dimensiune maximă a bufferului în număr de caractere

Verisuni (mai) sigure

StrSafe

```
HRESULT StringCchLength( LPCTSTR psz, size_t  cchMax, size_t  *pcch );

HRESULT StringCbLength( LPCTSTR psz, size_t  cbMax, size_t  *pcb );
```

Rezultatul: numărul de caractere

Dacă șirul depășește limitele – funcția eșuează cu **STRSAFE_E_INVALID_PARAMETER**

StrSafe

Exemple: afișare în string

Secured CRT

```c
/* wcsnlen_s example */
#include <stdio.h>
#include <stdlib.h> // for exit
#include <wchar.h>

#define MAX_PATH 20

int main() {
   wchar_t wsString[] = L"Hello World";
   size_t length;

   length = wcsnlen_s(wsString, MAX_PATH);

   wprintf_s(L"StringCchLength = %d \n", length);

   return 0;
}
```

Ce se întâmplă dacă MAX_PATH e mai mic decât lungimea șirului?

StrSafe

```c
/* StringCchLength example */
#include <stdio.h>
#include <stdlib.h> // for exit
#include <strsafe.h>

#define MAX_PATH 20

int main() {
   wchar_t wsString[] = L"Hello World";
   size_t length;
   HRESULT Res;

   Res = StringCchLength(wsString, MAX_PATH, &length);

   if (Res != S_OK) {
      wprintf_s(L"StringCchLength Failed: %s\n", wsString);
      exit(-1);
   }

   wprintf_s(L"StringCchLength = %d \n", length);
   return 0;
}
```

StringCchLength este o funcție generică

StrSafe

Afișare în șir de caractere

Verifică doar dacă șirul de formatare sau bufferul este NULL

CRT

```
int sprintf( char *buffer, const char *format [, argument] ... );

int swprintf( wchar_t *buffer, size_t count, const wchar_t *format [, argument]... );
```

Nu impune o limită maximă pentru dimensiune !

Secured CRT

```
int sprintf_s( char *buffer, size_t sizeOfBuffer, const char *format, ... );

int swprintf_s( wchar_t *buffer, size_t sizeOfBuffer, const wchar_t *format, ... );
```

Validează și specificatorii de format din șirul de formatare

Numărul maxim de caractere (octeți) care se vor stoca

Dacă bufferul este prea mic pentru textul formatat, atunci bufferul devine șirul vid și eșuează funcțialcu parametru invalid

StrSafe

```
HRESULT StringCchPrintf( LPTSTR  pszDest, size_t  cchDest, LPCTSTR pszFormat,  ... );

HRESULT StringCbPrintf( LPTSTR  pszDest, size_t  cbDest, LPCTSTR pszFormat,  ... );
```

Verisuni (mai) sigure

StrSafe

Exemple: preluare șir

Secured CRT

```
/* swprintf_s example */
#include <stdio.h>
#include <wchar.h>

int main() {
   wchar_t buf[100];
   int len = swprintf_s(buf, 100, L"%s", L"Hello world");
   wprintf_s(L"wrote %d characters\n", len);
   len = swprintf_s(buf, 100, L"%s", L"Hello\x0 world");
   wprintf_s(L"wrote %d characters\n", len);

   return 0;
}
```

Rezultat:

wrote 11 characters
wrote 5 characters

StrSafe

```
/* StringCchPrintf example */
#include <stdio.h>
#include <stdlib.h>
#include <strsafe.h>

#define BUF_SIZE 50

int main() {
   wchar_t msgBuf[BUF_SIZE];
   size_t cchStringSize;
   DWORD dwChars;

   int val1 = 10;
   int val2 = 20;

   StringCchPrintf(msgBuf, BUF_SIZE, L"Parameters = %d, %d",
                                          val1, val2);
   StringCchLength(msgBuf, BUF_SIZE, &cchStringSize);

   wprintf_s(L"length(%s) = %d \n",msgBuf, cchStringSize);
   return 0;
}
```

StringCchPrintf este o funcție generică

StrSafe

Character count functions	Byte count functions	Replaces
StringCchCat **StringCchCatEx**	**StringCbCat** **StringCbCatEx**	strcat, wcscat, _tcsat **lstrcat** **StrCat** **StrCatBuff**
StringCchCatN **StringCchCatNEx**	**StringCbCatN** **StringCbCatNEx**	strncat **StrNCat**
StringCchCopy **StringCchCopyEx**	**StringCbCopy** **StringCbCopyEx**	strcpy, wcscpy, _tcscpy **lstrcpy** **StrCpy**
StringCchCopyN **StringCchCopyNEx**	**StringCbCopyN** **StringCbCopyNEx**	strncpy, wcsncpy, _tcsncpy
StringCchGets **StringCchGetsEx**	**StringCbGets** **StringCbGetsEx**	gets, _getws, _getts
StringCchPrintf **StringCchPrintfEx**	**StringCbPrintf** **StringCbPrintfEx**	sprintf, swprintf, _stprintf **wsprintf** **wnsprintf** _snprintf, _snwprintf, _sntprintf
StringCchVPrintf **StringCchVPrintfEx**	**StringCbVPrintf** **StringCbVPrintfEx**	vsprintf, vswprintf, _vstprintf vsnprintf, _vsnwprintf, _vsntprintf **wvsprintf** **Wvnsprintf**
StringCchLength	**StringCbLength**	strlen, wcslen, _tcslen

Materiale de studiu

- Despre string-uri pe MSDN: https://msdn.microsoft.com/en-us/library/windows/desktop/ms646979(v=vs.85).aspx
- Despre categorii de funcții pe stringuri: http://zetcode.com/gui/winapi/strings/
- Johnson HART, Windows System Programming, 4th edition, Addison Wesley, 2010
  - Capitolul 1
- Jeffrey RICHTER & Christophe NASARRE, Windows via C, C++, 5th edition, Microsoft Press, 2008
  - Capitolul 2