Qwix – Einleitung

Qwix ist eine minimalistische (Makro-)Programmiersprache, die in C implementiert ist und NASM-win32 Code erzeugt.

1. Definitionsbefehle

|name par → Allokation im .bss-Segment
|name 'par' oder |name "par" → reserviert String
|name 0-9+ → reserviert Zahl
name. par → Variable im .data-Segment
'string' / "string" → String
{0-9+/Rechnung} → Integer (Redefinition nur bei Ints möglich)
Keine Floats, Doubles oder Arrays in Qwix
define name par → Makrodefinition (NASM %define)

2. Integrierte Befehle

print par ... → Konsolenausgabe (Strings).
markiert DWORD/Int → entspricht [par].
var randint min max → Zufallszahl (zur Compile-Zeit festgelegt).
Ausnahme: include [randint] erlaubt Runtime-Random.
atoi var string (var) → String → Int Konvertierung, Ergebnis in var.
prompt var msg → Eingabeaufforderung:
var muss vorher angelegt sein (db 128 dup(0)),
msg wird ausgegeben.
include name → Macro für extern
Mit [path] eigene Programme als .obj einbinden.

3. Makro-Befehle

clr num → Stack nach Funktionsaufruf leeren (add esp, num).
mov par par → Daten bewegen (# = dword).
xor par par → XOR.
try par par → Test (test par, par).

4. Zeichen-Operationen / Kontrollfluss

(.) var % var/num Compare → Vergleich (→ cmp).
Mit . vorne: nur Namen rechts/links übernehmen.
var & var → Stringvergleich (nur Variablen).
/ num → Return + Stack leeren.
* → Programmende (→ jmp exit8).
:name → Label (→ name:).
::name → Funktionsaufruf (→ call).
name.: → Jump-Definition (name:).
Sprung-Makros
! name → jne name (jump if not).
> name → jg name (jump if greater).

< name → jl name (jump if less).
? name → je name (jump if equal).

5. Weitere Zeichen & Operatoren

$/" " → Tabulator.
+par → Push auf den Stack (kein Wert → pop eax).
var ++ → Inkrement.
(.)# var value text reg → Variable für Qwix registrieren (z. B. wenn direkt in NASM deklariert).
; text → Kommentar.
~text(~) → Direktes NASM.
{0-9+/math} → Mathe-Ausdrücke (werden mit tinyexpr ausgewertet).

Beispiele

Simple MessageBox:

```
include '_MessageBoxA@16'
title. "Title"
msg. "Hello World!"
+0 +title +msg +0 ::'_MessageBoxA@16'
```

Simple random number:

```
include [randint] ; including it
randvar. {0} ; Var to load
randvar randint 1 10 ; Getting random num
fmt. "Random number: %d" ; Formated string
print #randvar fmt
```

Simple prompt + math:

```
password. "mypassword"
msg. "Enter Password: "
prompt. input msg
input & password ? secret*
access. "Access granted! Number is: %d"
secret.:
    a. {sqrt(3^2+4^2)}
    print #a access
```

Window Beispiel:

```
include '_GetModuleHandleA@4'
include '_RegisterClassExA@4'
include '_CreateWindowExA@48'
include '_ShowWindow@8'
include '_UpdateWindow@4'
include '_DefWindowProcA@16'
include '_GetMessageA@16'
include '_TranslateMessage@4'
include '_DispatchMessageA@4'
include '_PostQuitMessage@4'
include '_MessageBoxA@16'
include '_LoadIconA@8'
include '_LoadCursorA@8'
include '_LoadImageA@24'
include '_GetLastError@0'
include '_SendMessageA@16'
```

```
include '_sprintf'
include [lib\randint]
include [lib\morse]
include [lib\wincopy]

; Window
define WM_COMMAND '0x0111'
define WM_DESTROY '0x0002'
define WS_OVERLAPPEDWINDOW '0x00CF0000'
define WS_BORDER '0x00800000'
define ES_LEFT '0x0000'
define SW_SHOWNORMAL '1'

className. "WindowClass"
windowTitle. "You shall not pass"
|wc '48'
|msg '48'

; Button
define BUTTON_ID '1001'
define WS_CHILD '0x40000000'
define WS_VISIBLE '0x10000000'
define BS_PUSHBUTTON '0x00000000'

buttonClass. "BUTTON"
buttonText. "Enter"

; Edit
define WM_GETTEXT '0x000D'

editClass. "EDIT"
|hEdit 1
|editMsg 128

; Sources
iconPath. "Path\to\icon.ico"
cursorPath. "Path\to\cursor.cur"

; Secret
password. "qwix"
|secretMsg 64
secretFmt. "Magicnumber: %d"
magicNum. {0}

; Setup Window
+{0} ::'_GetModuleHandleA@4' mov esi eax
mov #[wc] 48
mov #[wc+4] 0
mov #[wc+8] wndProc
mov #[wc+12] 0
mov #[wc+16] 0
mov '[wc+20]' eax
+'0x10' +32 +32 +1 +iconPath +0 ::'_LoadImageA@24' ; Load icon
try eax eax ? checkE
mov #[wc+24] eax
mov #[wc+44] eax
+'0x10' +32 +32 +2 +cursorPath +0 ::'_LoadImageA@24' ; Load cursor
try eax eax ? checkE
mov #[wc+28] eax
mov #[wc+32] 6
mov #[wc+36] 0
mov #[wc+40] className

; Register class
+wc ::'_RegisterClassExA@4'
```

```
try eax eax ? checkE

; Create Window
+0 +esi +0 +0 +300 +400 +100 +100 +WS_OVERLAPPEDWINDOW
+windowTitle +className +0 ::'_CreateWindowExA@48' mov ebx eax

; Display window and update window
+'SW_SHOWNORMAL' +ebx ::'_ShowWindow@8'
+ebx ::'_UpdateWindow@4'

; --- Create Button ---
+0 +esi +BUTTON_ID +ebx +50 +50 +100 +100 +'WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON' +buttonText +but
::'_CreateWindowExA@48'
::'_CreateWindowExA@48'
; --- Create Edit ---
+0 +esi +0 +ebx +20 +200 +50 +50 +'WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT' +0 +editClass +0
::'_CreateWindowExA@48' mov '[hEdit]' eax
:msgloop

msgloop.:
    +0 +0 +0 +msg ::'_GetMessageA@16'
    try eax eax ? checkE
    +msg ::'_TranslateMessage@4'
    +msg ::'_DispatchMessageA@4'
    :msgloop

wndProc.:
    mov eax '[esp+8]'
    .eax % WM_COMMAND ? event
    .eax % WM_DESTROY ! defproc
    +0 ::'_PostQuitMessage@4'
    xor eax eax / 16

event.:
    mov eax '[esp+12]'
    .eax % BUTTON_ID ? buttonCmd ! defproc
    xor eax eax / 16

buttonCmd.:
    +editMsg +128 +WM_GETTEXT +#hEdit ::'_SendMessageA@16'
    editMsg & password ? secret
    +editMsg ::morse clr 4 mov ebx eax
    +0 +windowTitle +ebx +0 ::'_MessageBoxA@16'
    +ebx ::wincopy
    xor eax eax / 16

secret.:
    magicNum randint 1 10
    +#magicNum +secretFmt +secretMsg ::_sprintf clr 12
    +0 +windowTitle +secretMsg +0 ::'_MessageBoxA@16'

defproc.:
    mov eax '[esp+4]'
    mov ecx '[esp+8]'
    mov edx '[esp+12]'
    mov ebx '[esp+16]'
    +ebx +edx +ecx +eax
    ::'_DefWindowProcA@16' / 16

; ------- ERROR ---------
errTitle. "Error"
errFmt. "[Error] Code: %d"
|errCode 1
|errStr '12'
|errMsg 64
```

```
checkE.:
    ::'_GetLastError@0' mov '[errCode]' eax
    .eax % 0 ! error*

error.:
    print #errCode errFmt
    +#errCode +errFmt +errMsg ::_sprintf clr 12
    +'0x00000010' +errTitle +errMsg +0 ::'_MessageBoxA@16'*
```