QuadInt.dll
Anleitung

© by Dr. Jens-Arne Reumschüssel in 2024

Einführung:

QuadInt.dll ist eine 32bit-Bibliothek, die es ermöglicht, gewisse QuadInt-Funktionen auch in Programmiersprachen zu nutzen, die keine 8bit-Integer-Variablen kennen (z.B. XProfan vor Version X2). Dies ist u.a. für diverse API-Funktionen nützlich, die QuadInts als Parameter erwarten.

Es ist eine Include-Datei zur Nutzung mit XProfan beigefügt. Diese Anleitung ist für die Verwendung mit XProfan und der inc-Datei gedacht. In anderen Programmiersprachen müssen die Funktionen der DLL so aufgerufen werden, wie die Sprache es vorsieht. Insbesondere bei den Str-Funktionen ist dabei darauf zu achten, dass entsprechende Speicherbereich allokiert und genutzt werden, wobei ASCII-Strings zu verwenden sind. In XProfan nimmt einem die inc-Datei diese Arbeit ab.

Die DLL stellt beliebig viele QuadInt-Variablen zur Verfügung. Diese werden in den einzelnen Funktionen über eine logische Nummer angesprochen (z.B. 1 für das erste QuadInt, 2 für das zweite Quadint usw.). Mit diesen Variablen kann dann gerechnet werden, wobei die vier Grundrechenarten und Potenzieren enthalten sind. Auch die üblichen Vergleichsoperationen (gleich, kleiner, größer) sind implementiert.

Woran man sich gewöhnen muss: Sämtliche Funktionen bis auf fast alle Zuweisungs- und alle Auslesefunktionen arbeiten nicht mit Zahlen, sondern mit den Nummern der QuadInt-Variablen. QuadAdd(1,2,3) addiert nich etwas die Zahlen 2 und 3, sondern die Inhalte der QuadInt-Variablen mit den Nummern <2> und <3> und speichert das Ergebnis in der QuadInt-Variablen mit der Nummer <1>. Um die Unterscheidung zu erleichern, werden Nummern in dieser Anleitung immer in Klammern eingeschlossen.

Natürlich kann man für die Nummern der QuadInt-Variablen auch XProfan-Variablen benutzen, die etwas sprechender gestaltet werden können als reine Zahlen. Hierauf wurde im folgenden Bespiel der Übersicht halber allerdings verzichtet.

XProfan-Beispielprogramm:

```
$I QuadInt.inc

declare s$,hFile%,fn$,b#

cls
InitQuadInt(5) 'wir brauchen in diesem Programm 5 QuadInts

'eine Berechnung:
QuadLet(1,6) 'weist der QuadInt-Variablen <1> den Wert 6 zu
QuadLet(2,3) 'weist der QuadInt-Variablen <2> den Wert 3 zu
QuadPow(3,1,2) 'potenziert den Wert der QuadInt-Variablen <1> mit dem Exponenten <2> und weist <3> das Ergebnis zu; Lesart: <3> = <1>^<2> = 6^3
print "Pow="+str$(QuadGet(3)) 'holt das Ergebnis der Berechnung (hier direkt als LongInt, weil der Wert dieser Berechnung sicher in ein solches hineinpasst)
print "Lt="+str$(QuadLt(1,2)) 'ergibt 0, da der Wert von <1> (=6) größer ist als der Wert von <2> (=3)

'Nutzung mit API-Funktion:
```

```
fn$="QuadInt.inc"
dim b#,10000
hFile%=external("kernel32.dll","OpenFile",addr(fn$),b#,0) '0=~OF_READ
external("kernel32.dll","GetFileSizeEx",hFile%,QuadAddr(1)) 'übergibt die
Adresse der QuadInt-Variablen <1> an die API-Funktion, die ihr dann ein Er-
gebnis zuweist (so können auch sehr große Dateigrößen ermittelt werden)
external("kernel32.dll","CloseHandle",hFile%)
dispose b#
print "Dateigröße="+QuadGetStr(1)

'eine Schleife mit QuadInt-Laufvariable
QuadLetStr(1,"10000000001") 'Schleifenzähler
QuadLetStr(2,"10000000010") 'Schleifenende
QuadLet(3,1) 'Schrittgröße
QuadLet(4,4) 'Modulo-Divisor
print
while QuadLtOrEqu(1,2)
  QuadMod(5,1,4)
  print "Schleife: "+QuadGetStr(1)+" (Modulo="+QuadGetStr(5)+")"
  QuadAdd(1,1,3) 'aktuellen Zählerstand in <1> um 1 (ist in <3> gespei-
chert) erhöhen und Ergebnis wieder in <1> speichern; Lesart:
<1>=<1>+<3>=<1>+1
endwhile

'eine schnelle Schleife
QuadLetStr(1,"10000000001") 'Schleifenzähler
QuadLetStr(2,"10000000010") 'Schleifenende
QuadLet(3,1) 'Schrittgröße
QuadLet(4,4) 'Modulo-Divisor
print
QuadLoop(1,2,3,ProcAddr("CallbackProc",1),1)
ProcAddr("CallbackProc",-1)

PROC CallbackProc
  parameters par%
  QuadMod(5,1,4)
  print "Schleife: "+QuadGetStr(1)+" (Modulo="+QuadGetStr(5)+")"
ENDPROC 'CallbackProc

waitinput
DeInitQuadInt
end
```

Dll-Funktionsübersicht:

Initialisierung:

InitQuadInt(anz):
Vor Nutzung der DLL ist diese zu initialisieren. Dies geschieht mit dieser Funktion.
anz gibt die Anzahl der benötigten QuadInts an.
Achtung: Es wird in den übrigen Funktionen aus Performancegründen nicht überprüft, ob die Anzahl groß genug gewählt wurde!
Diese Funktion kann mehrfach aufgerufen werden, um die Anzahl der zur Verfügung stehenden QuadInt-Variablen anzupassen, wenn nötig.

DeInitQuadInt():
Muss nach der Nutzung der DLL aufgerufen werden. Diese Funktion hat keine Parameter.

Zuweisungen und Auslesungen:

QuadLet(<num>, value):
Weist der QuadInt-Variablen <num> den Wert value zu. Kann nur Werte bis zur maximalen Größe eines LongInt zuweisen, da die zugrundeliegende Programmiersprache keine QuadInts beherrscht. Für größere Werte ist QuadLetStr oder QuadLetFloat zu verwenden.

QuadLetStr(<num>, value$):
Weist der QuadInt-Variablen <num> den Wert val(value$) zu. Die zuzuweisende Zahl ist also als String zu übergeben. So kann einer QuadInt-Variablen ein echter QuadInt-Wert zugewiesen werden, auch wenn die benutzte Programmiersprache keine QuadInt-Zahlen beherrscht.

QuadLetFloat(<num>,fAddr):
Weist der QuadInt-Variablen <num> den ganzzahlig gerundeten Wert einer Float-Variablen (double precision) zu, deren Adresse in fAddr zu übergeben ist.

QuadLetQuad(<num1>, <num2>):
Weist der QuadInt-Variablen <num1> den Wert der QuadInt-Variablen <num2> zu.

QuadGet(<num>):
Liest den Wert der QuadInt-Variablen <num> aus. Kann nur Werte bis zur maximalen Größe eines LongInts lesen, da die zugrundeliegende Programmiersprache keine QuadInts beherrscht. Für größere Werte ist QuadGetStr oder QuadGetFloat zu verwenden.

QuadGetStr(<num>):
Liest den Wert der QuadInt-Variablen <num> als String aus.
Damit kann die zugrundeliegende Programmiersprache natürlich nichts weiter anfangen, als den Wert z.B. als Text auszugeben.

QuadGetFloat(fAddr,<num>):
Übergibt den Wert in der QuadInt-Variablen mit der Nummer <num> in eine Fließkommavariable (double precision), deren Adresse als erster Parameter zu übergeben ist.

Berechnungen:

QuadAdd(<erg>, <sum1>, <sum2>):
Die QuadInt-Variable <sum1> und die QuadInt-Variable <sum2> werden addiert.
Das Ergebnis wird der QuadInt-Variablen <erg> zugewiesen.
»Lesart«: erg=sum1+sum2

QuadSub(<erg>, <sub1>, <sub2>):
Die QuadInt-Variable <sub2> wird von der QuadInt-Variablen <sub1> abgezogen..
Das Ergebnis wird der QuadInt-Variablen <erg> zugewiesen.
»Lesart«: erg=sub1-sub2

QuadMul(<erg>, <mul1>, <mul2>):
Die QuadInt-Variable <mul1> und die QuadInt-Variable <mul2> werden multipliziert.
Das Ergebnis wird der QuadInt-Variablen <erg> zugewiesen.
»Lesart«: erg=mul1*mul2

QuadDivRound(art):
Mit dieser Funktion kann eingestellt werden, ob bei QuadDiv das Ergebnis gerundet werden soll (art=1), oder ob dessen Nachkommateil abgeschnitten werden soll (art=0, default).

QuadDiv(<erg>, <dividend>, <divisor>):
Die QuadInt-Variable <dividend> wird durch die QuadInt-Variable <divisor> geteilt.
Das Ergebnis wird der QuadInt-Variablen <erg> zugewiesen (Nachkommastellen werden abgeschnitten oder gerundet, je nach Einstellung in QuadDivRound).
»Lesart«: erg=dividend/divisor

QuadDiv2Float(fAddr, <dividend>, <divisor>):
Funktioniert genauso wie QuadDiv mit dem Unterschied, dass das Ergebnis in eine Fließkommavariable (double precision) gespeichert wird, deren Adresse als erster Parameter angegeben werden muss, anstatt in eine QuadInt-Variable.

QuadPow(<erg>, <basis>, <exp>):
Die QuadInt-Variable <basis> wird potenziert, wobei der Exponent dem Inhalt der QuadInt-Variablen <exp> entspricht.
Das Ergebnis wird der QuadInt-Variablen <erg> zugewiesen.
»Lesart«: erg=basis^exp

QuadMod(<erg>, <dividend>, <divisor>):
Der QuadInt-Variablen <erg> wird das Ergebnis der Modulo-Operation <dividend> mod <divisor> zugewiesen.
»Lesart«: erg=dividend mod divisor

Vergleiche:

QuadEqu(<num1>, <num2>):
gibt 1 zurück, wenn die Werte der QuadInt-Variablen <num1> und <num2> identisch sind, sonst 0.

QuadGt(<num1>, <num2>):
gibt 1 zurück, wenn der Wert der QuadInt-Variablen <num1> größer ist als der Wert der QuadInt-Variablen <num2>, sonst 0.

QuadGtOrEqu(<num1>, <num2>):
gibt 1 zurück, wenn der Wert der QuadInt-Variablen <num1> größer oder gleich dem Wert der QuadInt-Variablen <num2> ist, sonst 0.

QuadLt(<num1>, <num2>):
gibt 1 zurück, wenn der Wert der QuadInt-Variablen <num1> kleiner ist als der Wert der QuadInt-Variablen <num2>, sonst 0.

QuadLtOrEqu(<num1>, <num2>):
gibt 1 zurück, wenn der Wert der QuadInt-Variablen <num1> kleiner oder gleich dem Wert der QuadInt-Variablen <num2> ist, sonst 0.

Schleifen:

QuadLoop(<loop>, <stop>, <step>, CallbackAddress, UserValue):
Die QuadInt-Variable mit der Nummer <loop> muss vorher mit dem Startwert der Schleife gefüllt werden. Ebenso die QuadInt-Variable mit der Nummer <stop> mit dem Stop-Wert und diejenige mit der Nummer <step> mit dem Step-Wert.
Die Funktion ruft dann so lange die angegebene Callback-Funktion auf und erhöht bzw. erniedrigt jedes Mal die QuadInt-Variable mit der Nummer <loop> um den Wert in der QuadInt-Variablen mit der Nummer <step>, bis der Wert in der QuadInt-Variablen mit der Nummer <stop> erreicht ist.
Die Callback-Funktion muss genau einen Integer-Parameter haben. An diesen übergibt die Schleife bei jedem Aufruf den Wert, der in UserValue angegeben wurde.
In der Callback-Funktion kann dann alles, was man möchte, mit den QuadInt-Variablen angestellt werden (sogar die loop-, stop- und step-Variablen können verändert werden, wobei dahingestellt sei, ob das sinnvoll ist).
Schleifen können auch verschachtelt werden (d.h. in einer Callback-Funktion kann eine neue Schleife mit einer eigenen Callback-Funktion begonnen werden), sofern die benutzten Variablen hinreichend unterschiedlich sind (die Laufvariablen <loop> sollten z.B. niemals identisch sein).
Die maximale Verschachtelungstiefe ist zumindest bei Nutzung von XProfan auf 5 beschränkt, da @ProcAddr() nur fünf verschiedene Werte für Funktionen mit derselben Parameteranzahl zulässt.

QuadBreakLoop(<num>):
Bricht eine QuadLoop-Schleife vorzeitig ab.
<num> muss die Nummer der Loop-QuadInt-Variablen der Schleife sein (deren erster Parameter).

Adresse:

QuadAddr(<num>):
Gibt die Speicheradresse der QuadInt-Variablen <num> zurück.
Dabei gilt:
QuadAddr(<num>)=LowLongInt
QuadAddr(<num>)+4=HighLongInt

Version:

QuadGetVersion():
Gibt die Versionsnummer der DLL als String zurück. Auch hier muss vor dem Aufruf InitQuadInt aufgerufen worden sein.