Calling qfloat floating-point functions (LCC-WIN32) from Maple

As an example the cumulative normal distribution cdfN is chosen (given through the error function).

The library qfloat.dll should be in Window's system directory and the concurrent cdfn_mpl.dll should be places in the directory of this worksheet.

The C source for the DLL is given at the end.

AVt, Dec 2005

```
> restart;
  kernelopts(version);
```

Maple 10.02, IBM INTEL NT, Nov 8 2005 Build ID 208934

```
> Digits_lcc:=115;
```

Digits_lcc := 115

```
> Digits:=2*Digits_lcc; # greater precision to check results
```

Digits := 230

Define the cumulative normal distribution within Maple

```
> cdfN:= x -> 1/2+1/2*erf(1/2*x*2^(1/2));
```

$$\mathrm{cdfN} := x \to \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{1}{2}\,x\,\sqrt{2}\right)$$

For using the cdfn_mpl.dll locate its directory and call the external functions from there

```
> currentdir(): myDLL:=cat(%,`\\cdfn_mpl.dll`);
```

myDLL := "C:_Work\other\LCC_Work\cdfN_mpl\lcc\cdfn_mpl.dll"

A quick and dirty way is through strings:

```
> fct := define_external(
     'str_cdfN_str',
     'C',
     'x_str'::string[],
     'y_str'::string[],
     'nChar'::integer[4],
     RETURN::integer[4],
     LIB=myDLL):
```

This will provide the DLL with memory space (given as a string y_str) to store the results in the DLL. Since update is 'inplace' this will modify the string and its length. As Maple is a symbolic system one should never call this result directly, since this inconsistency for the same object will crash it (just try it and restart ...).

But a simple procedure solves the problem:

```
> cdfN_lcc:=proc(x)
     local X::string, Y;
     Y:=StringTools:-Fill( `0` , Digits_lcc);
     X:=convert(x,string);
     fct(X,Y,StringTools:-Length(Y));
     parse(Y);
  end proc:
```

Test that for inputs (first display yL from DLL, then Maple's result yM):

```
> xTst:= -1.2;
  yL:= cdfN_lcc(xTst):
  evalf(cdfN(xTst)): yM:=evalf(%,105):
  yL; yM;
  `absolute error`=yL-yM;
  `relative error`=evalf((yL-yM)/yM,105);
```

xTst := -1.2

0.1150696702217082680222202069566351486754470353375045415512633030900601152496502124010626\
    29851471926091647

0.1150696702217082680222202069566351486754470353375045415512633030900601152496502124010626\
    29851471926091647

absolute error = 0.

relative error = 0.

```
> xTst:= -10.2;
  yL:=cdfN_lcc(xTst):
  evalf(cdfN(xTst)): yM:=evalf(%,105):
  yL; yM;
  `absolute error`=yL-yM;
  `relative error`=evalf((yL-yM)/yM,105);
```

xTst := -10.2

0.9913625122559999052225802030899372106920272923794751452394434272410844535177121548944047\
    47882269785663070 10^{-24}

0.9913625122559999052225802030899372106920272923794751452394434272410844535177121549212947\
    08944570635892458 10^{-24}

absolute error = -0.26889961062300850229388 10^{-106}

relative error = -0.2712424640821706370883078156677152856207353409944865096678492849388640815 61\
    87140373997755002339813985040 7 10^{-82}

In the latter case one sees that qfloat exactness of 104 digits for the error function for those beyond the decimal point, while
Maple has a different notion for software floating-point numbers using
Float(SFloatMantissa(x), SFloatExponent(x)) = x .

```
> SFloatMantissa(yL),SFloatExponent(yL);
  SFloatMantissa(yM),SFloatExponent(yM);
```

99136251225599990522258020308993721069202729237947514523944342724108445351771215489440474 7\
    882269785663070, -129

99136251225599990522258020308993721069202729237947514523944342724108445351771215492129470 8\
    944570635892458, -129

One just has to accept this. So for small arguments the library returns 0:

```
> xTst:= -21.98;
  `lcc gives` = cdfN_lcc(xTst);
  `Maple says` = evalf(cdfN(xTst),200);
```

xTst := -21.98

lcc gives = 0.

Maple says = 0.2237309871609732229490043453727611228772453831271391761451608886367091038834 2\
    52240158209015783 10^{-106}

A more sound way of calling is to use byte arrays instead of strings on which the DLL should work.
The code in the DLL is the same, just the calling changes

```
> fct2 := define_external(
    'str_cdfN_str',
    'C',
    'x_str'::string[],
    'y_str'::ARRAY('datatype' = 'integer'[ 1 ], 'order' = 'C_order'),
    'nChar'::integer[4],
    RETURN::integer[4],
    LIB=myDLL):
```

In this case the byte array is just made a string through the function call and the function returns the length.

For simple use here is a procedure:

```
> cdfN_lcc2:=proc(x)
    local X::string, Y, YB;
    X:=convert(x,string);
    Y:=StringTools:-Fill( "0" , Digits_lcc );
    YB:=StringTools:-ToByteArray(Y);
    fct2(X,YB,StringTools:-Length(Y));
    StringTools:-FromByteArray(YB);
    parse(%);
  end proc:
```

and it gives the same result:

```
> xTst:= -1.2;
  cdfN_lcc2(xTst);
  cdfN_lcc(xTst);
```

```
                                        xTst := -1.2
0.1150696702217082680222202069566351486754470353375045415512633030900601152496502124010626\
    29851471926091647
0.1150696702217082680222202069566351486754470353375045415512633030900601152496502124010626\
    29851471926091647
```

A more direct but technical variant would be

```
> cdfN_lcc3:=proc(x)
    local X::string, YB, Filler, nChar;
    X:=convert(x,string);
    Filler:=48; # which is 0
    YB:=Array(1.. Digits_lcc + 10, 'datatype' = 'integer'[ 1 ],
      'order' = 'C_order',fill=Filler );
    nChar:= op(2,ArrayDims(YB));
    fct2(X,YB,nChar);
    StringTools:-FromByteArray(YB);
    parse(%);
  end proc:
```

which also gives the same result

```
> xTst:= -10.2;
  cdfN_lcc3(xTst);
  cdfN_lcc(xTst);
```

```
                              xTst := -10.2
0.991362512255999905222580203089937210692027292379475145239443427241084453517712154894404 7\
    47882269785663070 10^(-24)
0.9913625122559999052225802030899372106920272923794751452394434272410844535177121548944047\
    47882269785663070 10^(-24)
```

I have not tried to use a Maple generated wrapper to access the DLL as the above already works ...

Finally look how one could extend the function for large arguments using asymtotics, which is coded in the following function

```
> fctplus := define_external(
    'str_cdfNplus_str',
    'C',
    'x_str'::string[],
    'y_str'::string[],
    'nChar'::integer[4],
    RETURN::integer[4],
    LIB=myDLL):

  cdfNplus_lcc:=proc(x)
    local X::string, Y;
    Y:=StringTools:-Fill( `0` , Digits_lcc + 10 );
    X:=convert(x,string);
    fctplus(X,Y,StringTools:-Length(Y));
    parse(Y);
  end proc:
```

Switch to high precision (say: 2000 digits)

```
> remDigits := Digits:
  Digits := 2000;
                              Digits := 2000
> xTst:= -30.2;
  yL:=cdfNplus_lcc(xTst):
  evalf(cdfN(xTst)): yM:=evalf(%,105):
  yL; yM; #yL-yM;
                              xTst := -30.2
0.1184288649716832304264669071633940892295166490180330791061169865458047886309556567410733\
    89138834722621161 10^(-199)
0.1184291478610912490141903934125399685949521935821888239243869608754971508624578423303429\
    46746868292907314 10^(-199)
> xTst:= -90.12345678901234567890123456789012345678901234567890;
  yL:=cdfNplus_lcc(xTst):
  evalf(cdfN(xTst)): yM:=evalf(%,105):
  yL; yM;
              xTst := -90.12345678901234567890123456789012345678901234567890
0.8405492852286168915719447824748309966618559949642607627963973161998674489615437101999151\
    18077834654341891 10^(-1766)
0.8405493106922971678309574831392146928478196984537390475050634078444208655500997471018369\
    66454436335733862 10^(-1766)
```

The leading zeros are correct and then the next 5 - 6 digits and due to the rude solution I used.

```
> Digits:=remDigits:
>
```

Sources for cdfn_mpl.dll

Export the functions explicitely through a .def file

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

BOOL WINAPI __declspec(dllexport) LibMain(HINSTANCE hDLLInst, DWORD
fdwReason, LPVOID lpvReserved)
{
  switch (fdwReason)
  {
    case DLL_PROCESS_ATTACH:
      break;
    case DLL_PROCESS_DETACH:
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
  }
  return TRUE;
}

extern __declspec(dllexport) long __stdcall
str_cdfN_str (char* x_str, char* y_str, int nChar)
{
 qfloat result;
 qfloat X;

 asctoq(x_str,&X);
 result = (erfq(X/sqrtq(2.0q)) + 1.0q)/2.0q;
 qtoasc(&result, y_str, min(strlen(y_str), 115));
 return strlen(y_str);
}


qfloat asymptotic_cdfN(qfloat X)
{
 qfloat c; // 1/sqrt(2*Pi)
 qfloat pdfN_X;
 qfloat result;

 if ( X == 0.0q )
  {return 0.0q;}
 c = "0.398942280401432677939946059934381868475858631164934657665925829670\
0657925899301838501252333907306936430302 6";
 pdfN_X = expq( - X*X / 2.0q ) * c;
 result = - X / (1.0q + X*X) * pdfN_X;

 if ( 0.0q < X )
  {return result = 1.0q + result;}
 return result;
}


extern __declspec(dllexport) long __stdcall
```

```
str_cdfNplus_str (char* x_str, char* y_str, int nChar)
{
 qfloat result;
 qfloat X;

 asctoq(x_str,&X);
 if ( 21.97q < abs(X) )
  { result = asymptotic_cdfN(X);}
 else
  { result = (erfq(X/sqrtq(2.0q)) + 1.0q)/2.0q;}

 qtoasc(&result, y_str, min(strlen(y_str), 115));
 return strlen(y_str);
}
```