Using the qfloat floating-point library from LCC-WIN32 through Maple

As an example the cumulative normal distribution cdfN is chosen (given through the error function).
A method is shown how to work with qfloat functions in the system library qflaot.dll using their names.

The library `qfloat.dll` should be in Window's system directory and the concurrent `lcc_mpl.dll` should be placed in the directory of this worksheet.

The C sources for the DLL are given at the end.

AVt, Dec 2005

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

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

```
> Digits_lcc:=105;
  Digits:=2*Digits_lcc; # greater precision to check results
```

Digits_lcc := 105

Digits := 210

Calling qfloat function coded in a DLL (compiled with LCC)

For using the `cdfn_mpl.dll` locate its directory and call the external functions from there (see the appendix for the C code within the DLL):

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

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

As an example the cumulative normal distribution is treadted, given within Maple as

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

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

A quick and dirty way to access functions 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+10);
    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);
```

$$xTst := -1.2$$

$$0.11506967022170826802222020695663514867544703533750454155126330309006011524965021240\backslash 106262985147192609164 7$$

$$0.11506967022170826802222020695663514867544703533750454155126330309006011524965021240\backslash 106262985147192609164 7$$

$$\text{absolute error} = 0.$$

$$\text{relative error} = 0.$$

```
> 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);
```

$$xTst := -10.2$$

$$0.99136251225599990522258020308993721069202729237947514523944342724108445351771215489\backslash 44047478822697856630 70\ 10^{-24}$$

$$0.99136251225599990522258020308993721069202729237947514523944342724108445351771215492\backslash 12947089445706358924 58\ 10^{-24}$$

$$\text{absolute error} = -0.26889961062300850229388\ 10^{-106}$$

$$\text{relative error} = -0.27124246408217063708830781566771528562073534099448650966784928493886408\backslash 15618714037399775500233981398504 07\ 10^{-82}$$

In the latter case one sees that qfloat understands the exactness of 104 digits 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);
```

$$9913625122559999052225802030899372106920272923794751452394434272410844535177121548944\backslash 0\ 4747882269785663070, -129$$

$$9913625122559999052225802030899372106920272923794751452394434272410844535177121549212\backslash 9\ 4708944570635892458, -129$$

One just has to accept this (or has to code some other solution). In general qfloat does not restrict to digits after the decimal point, see below. But here 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 = 0.22373098716097322294900434537276112287724538312713917614516088863670910\
38342522401582090157 83 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+10 );
    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

0.11506967022170826802222020695663514867544703533750454155126330309006011524965021240 1\
062629851471926091647

0.11506967022170826802222020695663514867544703533750454155126330309006011524965021240 1\
062629851471926091647

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);
```

$$xTst := -10.2$$

$$0.99136251225599990522258020308993721069202729237947514523944342724108445351771215489 4\backslash$$
$$04074788226978566307010^{-24}$$

$$0.99136251225599990522258020308993721069202729237947514523944342724108445351771215489 4\backslash$$
$$04074788226978566307010^{-24}$$

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

Calling functions from system library qfloat.dll

This is the main library to work with qfloat numbers. It has to be located in Window's system directory.

All the functions in that library can be seen by using "`pedump qfloat.dll`" at command line level in the
directory `..\lcc\bin`. The functions start with an underscore and usually end with suffix q.

For proper use one should consult the documentation and check the current prototypes in `qfloat.h`, as the download is more up to date.

For using these function a wrapper is convenient and to have a flexible solution I prefer some which will accept function through their names (as strings). The underscore will be provided through Maple, so the names are as they are in the documentation.

I just take the simpliest case: input and output are 1-dimensional real numeric and of datatype qfloat.

```
> qFct_DLL := define_external(
    'str_qFct_str',
    'C',
    'x_str'::string[],
    'y_str'::string[],
    'fctName_str'::string[],
    RETURN::integer[4],
    LIB=myDLL):

  qFct:=proc(x,fctName::string)
    local X::string, Y, F::string, result;

    Y:= StringTools:-Fill( `0` , Digits_lcc+10);
    X:= convert(x,string);
    F:= cat("_",fctName);

    result:=qFct_DLL(X,Y,F);

    if result = - 1 then
      return "qfloat.dll not found";
    elif result = - 2 then
      return cat("function ", fct, " not found in qfloat.dll");
    end if;

    return parse(Y);
  end proc:
> 
```

```
> fctTst:= "sinq";
  xTst:= 0.5;

  qFct(xTst, fctTst);
  sin(xTst): evalf(%,105);
```

fctTst := "sinq"

xTst := 0.5

0.479425538604203000273287935215571388081803367940600675188616613125535000287814832209\
63127468434826908613 2

0.479425538604203000273287935215571388081803367940600675188616613125535000287814832209\
63127468434826908613 2

Note that calling the DLL with symbolics will not work, so for some x = $\frac{\pi}{4}$ one has to evalute to numerics first:

```
> fctTst:= "sinq";
  xTst:= evalf(Pi/4,105);

  qFct(xTst, fctTst);
  sin(xTst): evalf(%,105);
```

fctTst := "sinq"

xTst := 0.785398163397448309615660845819875721049292349843776455243736148076954101571552\
24965700870633552926699553 8

0.707106781186547524400844362104849039284835937688474036588339868995366239231053519425\
193767163820786367502

0.707106781186547524400844362104849039284835937688474036588339868995366239231053519425\
193767163820786367508

As already stated above qfloat does not restrict to 104 digits after the decimal point, it has this exactness in
the usual sense:

```
> fctTst:= "expq";
  xTst:= -100^2;

  qFct(xTst, fctTst);
  exp(xTst): evalf(%,105);
```

fctTst := "expq"

xTst := -10000

0.113548386531473609854093887506624840195743161009031884267155265915730430241273915784\
077651790431252146122 10^{-4342}

0.113548386531473609854093887506624840195743161009031884267155265915730430241273915784\
077651790431252146122 10^{-4342}

Note that the simple wrapper performs no type checking and of course does not check for valid ranges.

The library qfloat provides complex numerics (through structures). As this currently does not cover high
precision I have not coded the wrapper for this types.

Appendix 1: Sources for cdfn_mpl.dll

Compile it as multi-user and the option "without underscores".

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);
}


// accessing qfloat.dll

typedef qfloat (* q_fct_q)(qfloat*);
q_fct_q myfct;


qfloat
call_qfloat(qfloat* x, char* fctName, int* errCode)
{
 char * theLIB;
 qfloat result;

 theLIB = "qfloat.dll";

 HINSTANCE hLib=LoadLibrary(theLIB);

 if(hLib==NULL) {
  FreeLibrary((HMODULE)hLib);
  *errCode = 1;
  return 0.0q;
```

```
  }


 myfct = (q_fct_q)GetProcAddress((HMODULE)hLib, fctName);

 if((myfct==NULL)) {
  FreeLibrary((HMODULE)hLib);
  *errCode = 2;
  return 0.0q;
 }

 result = myfct(x);

 FreeLibrary((HMODULE)hLib);

 return result;
}


extern __declspec(dllexport) long __stdcall
str_qFct_str (char* x_str, char* y_str, char* fctName)
{
 qfloat result;
 qfloat X;
 int errCode;

 asctoq(x_str,&X);

 result = call_qfloat(&X, fctName, &errCode);

 if ( 0 < errCode){
  return - errCode;}

 qtoasc(&result, y_str, min(strlen(y_str), 115));

 return strlen(y_str);
}
```

Appendix 2: Contents of qfloat.dll

```
Entry Pt  Ordn  Name
00003197     1  _LibMain@12
00002B7B     2  ____cacos_pstruct_qfloatstruct__Complex
00002BB4     3  ____cacosh_pstruct_qfloatstruct__Complex
00002B8E     4  ____casin_pstruct_qfloatstruct__Complex
00002BA1     5  ____casinh_pstruct_qfloatstruct__Complex
00002780     6  ____cbrt_pqfloat
00002793     7  ____copysign_pqfloat_pqfloat
000022F5     8  ____fabs_pqfloat
0000260C     9  ____floor_pqfloat
000024C5    10  ____log_pqfloat
0000273F    11  ____pow_pqfloat_pqfloat
000023CA    12  ____sqrt_pqfloat
00002184    13  __op_asgn_pdouble_pqfloat
000021C0    14  __op_asgn_pint_pqfloat
000020A7    15  __op_asgn_pqfloat_double
000020E0    16  __op_asgn_pqfloat_float
00002072    17  __op_asgn_pqfloat_int
00002121    18  __op_asgn_pqfloat_long_double
00002160    19  __op_asgn_pqfloat_pchar
00002BC7    20  __op_asgn_pstruct_qfloatstruct__Complex_int
00002BF8    21  __op_asgn_pstruct_qfloatstruct__Complex_long_double
```

```
00002C29    22
__op_asgn_pstruct_qfloatstruct__Complex_pstruct_long_double__Complex
00002B0B    23
__op_asgn_pstruct_qfloatstruct__Complex_struct_long_double__Complex
0000221E    24  __op_cast_double_qfloat
000021E9    25  __op_cast_int_qfloat
00002232    26  __op_cast_long_double_qfloat
00002246    27  __op_cast_longlong_qfloat
000022B0    28  __op_cast_pqfloat_double
0000227B    29  __op_cast_pqfloat_int
000021A6    30  __op_cast_pqfloat_long_double
000022C9    31  __op_cast_pqfloat_longlong
00002299    32  __op_cast_pqfloat_unsigned_int
00002AD2    33
__op_cast_struct_long_double__Complex_struct_qfloatstruct__Complex
0000225A    34  __op_cast_unsigned_int_qfloat
000021FD    35  __op_cast_unsigned_long_qfloat
00001A6A    36  __op_decrement_pqfloat
00001B2B    37  __op_divasgn_pqfloat_pqfloat
00001B18    38  __op_divide_pqfloat_pqfloat
00002ABC    39
__op_divide_pstruct_qfloatstruct__Complex_pstruct_qfloatstruct__Complex
00001B99    40  __op_equal_double_pqfloat
00001B6B    41  __op_equal_pqfloat_double
00001BC7    42  __op_equal_pqfloat_int
00001B4E    43  __op_equal_pqfloat_pqfloat
00001FAC    44  __op_greater_double_pqfloat
00001FDB    45  __op_greater_int_pqfloat
00001F7D    46  __op_greater_pqfloat_double
00001F4E    47  __op_greater_pqfloat_int
00001F31    48  __op_greater_pqfloat_pqfloat
00002027    49  __op_greaterequal_pqfloat_int
0000200A    50  __op_greaterequal_pqfloat_pqfloat
00001A45    51  __op_increment_pqfloat
00001EDB    52  __op_less_double_pqfloat
00001E37    53  __op_less_int_pqfloat
00001DE1    54  __op_less_pqfloat_double
00001E89    55  __op_less_pqfloat_int
00001DC3    56  __op_less_pqfloat_pqfloat
00002055    57  __op_lessequal_pqfloat_pqfloat
00001A8F    58  __op_minus_pqfloat
00001A1D    59  __op_minus_pqfloat_pqfloat
00002A90    60
__op_minus_pstruct_qfloatstruct__Complex_pstruct_qfloatstruct__Complex
00001ABF    61  __op_minusasgn_pqfloat_pqfloat
00001AF5    62  __op_multasgn_pqfloat_pqfloat
00001AE2    63  __op_multiply_pqfloat_pqfloat
00002AA6    64
__op_multiply_pstruct_qfloatstruct__Complex_pstruct_qfloatstruct__Complex
00001D1F    65  __op_notEqual_double_pqfloat
00001C98    66  __op_notEqual_pqfloat_double
00001D73    67  __op_notEqual_pqfloat_int
00001C12    68  __op_notEqual_pqfloat_long_double
00001BF5    69  __op_notEqual_pqfloat_pqfloat
00001A0A    70  __op_plus_pqfloat_pqfloat
00002A7A    71
__op_plus_pstruct_qfloatstruct__Complex_pstruct_qfloatstruct__Complex
000019BD    72  __op_plusasgn_plong_double_pqfloat
000019E1    73  __op_plusasgn_pqfloat_double
00001994    74  __op_plusasgn_pqfloat_long_double
0000197F    75  __op_plusasgn_pqfloat_pqfloat
000027C6    76  _acoshq
```

```
00002581    77  _acosq
00008FBC    78  _airyq
0000F20A    79  _asctoq
000027D9    80  _asinhq
0000256E    81  _asinq
000027EC    82  _atanhq
00002548    83  _atanq
00002E81    84  _beta_incompleteq
00002DD0    85  _betaq
00002C60    86  _cabsq
00002BB4    87  _cacoshq
00002B7B    88  _cacosq
00002BA1    89  _casinhq
00002B8E    90  _casinq
00002CF8    91  _catanhq
00002CAC    92  _catanq
00002780    93  _cbrtq
00002CBF    94  _ccoshq
00002C86    95  _ccosq
000026F9    96  _ceilq
00002D0B    97  _cexpq
00002D1E    98  _clogq
00002793    99  _copysignq
00002812   100  _coshq
0000255B   101  _cosq
00002D31   102  _cpowq
00002CD2   103  _csinhq
00002C73   104  _csinq
00002D47   105  _csqrtq
00002CE5   106  _ctanhq
00002C99   107  _ctanq
0000EAC5   108  _e113toq
0000E336   109  _e24toq
0000E82E   110  _e64toq
00002E9A   111  _ellipticKq
00002905   112  _erfcq
000028F2   113  _erfq
0000E582   114  _etoq
000023F0   115  _exp2q
000028CC   116  _expm1q
000023DD   117  _expq
00002325   118  _fabsq
00002344   119  _fdimq
00002634   120  _floorq
00002985   121  _fmaxq
000029C0   122  _fminq
000029FB   123  _fmodpiq
000022DF   124  _fmodq
0000284E   125  _frexpq
00002ED4   126  _hypergeom2f1q
00002E29   127  _hypergeomq
00002A3C   128  _hypotq
0000293E   129  _igammaq
00002864   130  _ilogbq
00003D20   131  _initqgam
0000F9F3   132  _isfiniteq
0000F775   133  _isinfq
00002D5A   134  _j0q
00002D72   135  _j1q
00002EAD   136  _jnq
000028B6   137  _ldexpq
0000BD3E   138  _lgam
```

```
00002918   139  _lgammaq
0000F645   140  _lltoq
000024ED   141  _log10q
000028DF   142  _log1pq
0000250F   143  _log2q
000027B3   144  _logbq
000024B2   145  _logq
0000D856   146  _ltoq
00008D34   147  _mtherr
0000CAF0   148  _ndtri
0001B458   149  _oneopi
0000276A   150  _powq
00002E42   151  _psiq
0001B3B0   152  _q32
0000D834   153  _qabs
00001525   154  _qacos
00001324   155  _qacosh
0000DABA   156  _qadd@12
000013B8   157  _qasin
000032DC   158  _qasinh
000031A0   159  _qatanh
0000159C   160  _qatn
000017CC   161  _qatn2
000078BC   162  _qcabs
00007FED   163  _qcacos
00008631   164  _qcacosh
00007068   165  _qcadd
00007C43   166  _qcasin
00008538   167  _qcasinh
0000825B   168  _qcatan
00008713   169  _qcatanh
000038FC   170  _qcbrt
00007B73   171  _qccos
000085A4   172  _qccosh
00008103   173  _qccot
0000748A   174  _qcdiv
000079B8   175  _qcexp
00010448   176  _qclear@4
00007A60   177  _qclog
0000787C   178  _qcmov
0000F15A   179  _qcmp
000070CE   180  _qcmul
000078A6   181  _qcneg
0001D8B4   182  _qcone
0000F787   183  _qcopysign
0000374C   184  _qcosh
00008BC8   185  _qcot
0000877F   186  _qcpow
00007AB5   187  _qcsin
000084B3   188  _qcsinh
00007D36   189  _qcsqrt
0000709B   190  _qcsub
0000804E   191  _qctan
0000866C   192  _qctanh
0001D844   193  _qczero
0000DD16   194  _qdiv@12
0000D5D8   195  _qellpk
00004BA0   196  _qerf
00004DE0   197  _qerfc
0001B4C8   198  _qeul
000049E8   199  _qexp10
00004A10   200  _qexp2
```

```
00004A60   201  _qfact
00003710   202  _qfcos
00004868   203  _qfexp
00006880   204  _qflog
00008BF0   205  _qfloor
00006C94   206  _qfpow
0000DA58   207  _qfrexp
00008838   208  _qfsin
0000D710   209  _qfsqrt
00008A50   210  _qftan
0001BF28   211  _qgam12
00019D08   212  _qgamcof
0001BF60   213  _qgamini
0000409A   214  _qgamma
0001B298   215  _qhalf
0000D951   216  _qifrac
00004473   217  _qigam
0000418C   218  _qigamc
0000CCCC   219  _qincb
0000C598   220  _qincbi
0000F7A6   221  _qincr
0000E2D4   222  _qinfin
00002838   223  _qipow
000051B8   224  _qjn
0000DA05   225  _qldexp
00003DD5   226  _qlgam
00006C68   227  _qlog10
0001B3E8   228  _qlog2
00004A3A   229  _qlogtwo
0001B228   230  _qminusone
00010410   231  _qmov
00010428   232  _qmovz
0001B810   233  _qmtens
0000DE02   234  _qmul@12
0000DF82   235  _qmuli
0000C338   236  _qndtr
0000C084   237  _qndtri
0000D84C   238  _qneg
0001B378   239  _qnine
0001B2D0   240  _qone
0001B490   241  _qpi
00006D4C   242  _qpowi
00002393   243  _qprint
00004681   244  _qpsi
000025FE   245  _qrand
00002594   246  _qratio
000025C9   247  _qratiol
000081C3   248  _qredpi
00006F10   249  _qremain
00003BE0   250  _qsinh
0001B420   251  _qsqrt2
0000DA9C   252  _qsub@12
0000378C   253  _qtanh
0001B500   254  _qtens
0001B340   255  _qthree
0000EE42   256  _qtoasc
0000E65C   257  _qtoe
0000EBDA   258  _qtoe113
0000E436   259  _qtoe24
0000E905   260  _qtoe64
00001934   261  _qtof
000018F8   262  _qtoi
```

```
00001916   263  _qtol
0001B308   264  _qtwo
00005EDC   265  _qyn
0001B260   266  _qzero
00002954   267  _roundq
00002967   268  _scalbnq
00002A35   269  _signbitq
000027FF   270  _sinhq
00002522   271  _sinq
000023CA   272  _sqrtq
00002E6B   273  _students_t_invq
00002E55   274  _students_tq
00002825   275  _tanhq
00002535   276  _tanq
0000292B   277  _tgammaq
00002D8A   278  _y0q
00002DA2   279  _y1q
00002DBA   280  _ynq
```