WG 14 N1723

Encoding interchange formats in arrays

The WG 14 committee in Delft questioned the need for C types to support the IEC 60559 non-arithmetic interchange formats and recent email from Joseph Meyers suggested using arrays instead. Whence the following approach.

Re-encoding functions

The re-encoding functions below are intended to facilitate interchange of decimal data encoded in either of the decimal encoding schemes specified in IEC 60559. If a program on one implementation uses an encoding function to encode a **`_Decimal`***N* value **`x`** and uses **`fwrite`** to write the resulting encoding to a file, then another program on another implementation can use **`fread`** to read the encoding from the file, correct the byte order if necessary (by usual means), call the decoding function for the given encoding scheme, and thereby obtain the exact value **`x`** written by the first program, provided the implementations have the usual 8-bit bytes.

`void decodedecd`*N* **`(_Decimal`***N* **`* restrict xptr, const unsigned char * restrict encptr);`**

These functions interpret the *N*/8 element array pointed to by **`encptr`** as an IEC 60559 decimal*N* encoding in the encoding scheme based on decimal encoding of the significand, convert the given encoding into a representation in the type **`_Decimal`***N*, and store the result in the object pointed to by **`xptr`**. These functions preserve the encoded value and raise no floating-point exceptions. If the encoding is non-canonical, these functions may or may not produce a canonical representation.

`void encodedecd`*N* **`(unsigned char * restrict encptr, _Decimal`***N* **`* restrict xptr);`**

These functions convert **`*xptr`** into an IEC 60559 decimal*N* encoding in the encoding scheme based on decimal encoding of the significand and store the resulting encoding as an *N*/8 element array in the object pointed to by **`encptr`**. These functions preserve the value of **`*xptr`** and raise no floating-point exceptions. If **`*xptr`** is non-canonical, these functions may or may not produce a canonical encoding.

`void decodebind`*N* **`(_Decimal`***N* **`* restrict xptr, const unsigned char * restrict encptr);`**

These functions interpret the *N*/8 element array pointed to by **`encptr`** as an IEC 60559 decimal*N* encoding in the encoding scheme based on binary encoding of the significand, convert the given encoding into a representation in the type **`_Decimal`***N*, and store the result in the object pointed to by **`xptr`**. These functions preserve the encoded value and raise no floating-point exceptions. If the encoding is non-canonical, these functions may or may not produce a canonical representation.

`void encodebind`*N* **`(unsigned char * restrict encptr, _Decimal`***N* **`* restrict xptr);`**

These functions convert **`*xptr`** into an IEC 60559 decima*N* encoding in the encoding scheme based on binary encoding of the significand and store the resulting encoding as an *N*/8 element array in the object pointed to by **`encptr`**. These functions preserve the value of *xptr and raise no floating-point exceptions. If **`*xptr`** is non-canonical, these functions may or may not produce a canonical encoding.

The following functions are intended to support interchange of binary data in the encoding specified by IEC 60559.

```
void decodefN (_FloatN * restrict xptr, const unsigned char * restrict
encptr);
```

These functions interpret the *N*/8 element array pointed to by **`encptr`** as an IEC 60559 binary*N* encoding, convert the given encoding into a representation in the type **`_Float`***N*, and store the result in the object pointed to by **`xptr`**. These functions preserve the encoded value and raise no floating-point exceptions. If the encoding is non-canonical, these functions may or may not produce a canonical representation.

```
void encodefN (unsigned char * restrict encptr, _FloatN * restrict
xptr);
```

These function convert **`*xptr`** into an IEC 60559 binary*N* encoding and store the resulting encoding as an *N*/8 element array in the object pointed to by **`encptr`**. These functions preserve the value of **`*xptr`** and raise no floating-point exceptions. If **`*xptr`** is non-canonical, these functions may or may not produce a canonical encoding.

Encoding conversion functions

An implementation may support IEC 60559 non-arithmetic formats by providing, for each decimal format of width *N*,

```
void strfromdecencdN (char * restrict s, size_t n, const char *
restrict format, const unsigned char * restrict encptr);

void strtodecencdN (unsigned char * restrict encptr, const char *
restrict nptr, char ** restrict endptr);

void strfrombinencdN (char * restrict s, size_t n, const char *
restrict format, const unsigned char * restrict enc);

void strtobinencdN (unsigned char * restrict encptr, const char *
restrict nptr, char ** restrict endptr);
```

and, for each binary format of width *N*,

```
void strfromencdN (char * restrict s, size_t n, const char * restrict
format, const unsigned char * restrict enc);

void strtoencdN (unsigned char * restrict encptr, const char * restrict
nptr, char ** restrict endptr);
```

which convert, as specified in IEC 60559, between character sequences and arrays containing encodings.

Note that **`A`**, **`a`** style formatting can produce exact conversions. Thus, these (strfrom and strto) functions, in combination with other conversion operations, provide (execution-time) initialization of encodings and conversions between encodings and other encodings, real floating types supporting IEC 60559 formats, and character sequences, which are functionalities that IEC 60559 requires for all its supported formats (including non-arithmetic formats). However, the performance cost of two character sequence conversions might be prohibitive, so that more functions will be needed to provide the requisite conversions more directly. Conversions between all supported widths for each encoding scheme should be adequate:

for decimal,

```
void dMdecencdN (unsigned char * restrict encM, const unsigned char *
restrict encN);
```

```
void dMbinencdN (unsigned char * restrict encM, const unsigned char *
restrict encN);
```

and for binary,

```
void fMencfN (unsigned char * restrict encM, const unsigned char *
restrict encN);
```

Jim Thomas
2013-06-10