C++FA 2.31 LAB 8: FAST Example

This simple example, developed by Brian Schneider, shows how to parse the FAST message presented in the text. This example fits in section C++FA 2.31 of ***C++ for Financial Applications*** by Ben Van Vliet, available at **www.benvanvliet.com**.

FAST_Example.cpp

```
#include <iostream>
using namespace std;

struct Trade
{
  char Symbol[5];
  char BS;
  double Price;
  unsigned int Qty;
};

int main()
{
  // FAST Message
  unsigned int msg[] = { 0xf8, 0x80, 0x47, 0xcd, 0x16, 0xd4, 0x7e, 0x0d, 0x84 };

  Trade m_Trade;
  int CurrByte = 0; // Current byte
  int exp = 0;

  for( unsigned int j = 128; j > 0; j /= 2 )
  {
    switch( j )
    {
      case 128: // Check PMAP
        if( msg[ 0 ] & j ) // Confirm SBit and increment CurrByte to next field
        {
          ++CurrByte;
        }
        else
        {
          cout << "Invalid FAST Message: PMAP field must be first byte" << endl;
        }
        break;

      case 64:
        if( msg[ 0 ] & j ) // Check Buy/Sell
        {
          m_Trade.BS = ( msg[ CurrByte ] & 1)  ? 'B' : 'S';
          if( msg[ CurrByte ] & 128 )
          // Confirm SBit for Buy/Sell and increment CurrByte
          {
            ++CurrByte;
          }
          else
          {
```

```
            cout << "Invalid message: Buy/Sell field longer than 1 byte" << endl;
            break;
          }
        }
        break;

      case 32:  // Check Symbol (allow for size of 1 to 4 characters)
        if( msg[ 0 ] & j )
        {
          m_Trade.Symbol[ 4 ] = 0x0;
          bool decode_complete = false;
          for( int i = 0; i < 4; i++ )
          {
            switch( i )
            {
              case 0:
                if( msg[ CurrByte ] & 128 ) // If stop bit set, decode and complete
                {
                  m_Trade.Symbol[ i ] = ( char )( msg[ CurrByte ] & 127 );
                  decode_complete = true;
                }
                else // Decode and increment
                {
                  m_Trade.Symbol[ i ] = ( char )( msg[ CurrByte ] & 127 );
                  ++CurrByte;
                }
                break;
              case 1:
                if( decode_complete == true )
                {
                  m_Trade.Symbol[ i ] = 0x0;
                }  // Zero fill
                else
                {
                  if( msg[ CurrByte ] & 128 )
                  {
                    m_Trade.Symbol[ i ] = ( char )( msg[ CurrByte ] & 127 );
                    decode_complete = true;
                  }
                  else
                  {
                    m_Trade.Symbol[ i ] = ( char )( msg[ CurrByte ] & 127 );
                    ++CurrByte;
                  }
                }
                break;
              case 2:
                if( decode_complete == true )
                {
                  m_Trade.Symbol[ i ] = 0x0;
                }  // Zero fill
                else
                {
                  if( msg[ CurrByte ] & 128 )
                  {
                    m_Trade.Symbol[ i ] = ( char )( msg[ CurrByte ] & 127 );
                    decode_complete = true;
```

```
                }
                else
                {
                  m_Trade.Symbol[ i ] = ( char )( msg[ CurrByte ] & 127 );
                  ++CurrByte;
                }
              }
              break;
            case 3:
              if( decode_complete == true )
              {
                m_Trade.Symbol[ i ] = 0x0;
              }  // Zero fill
              else
              {
                if( msg[ CurrByte ] & 128 )
                {
                  m_Trade.Symbol[ i ] = ( char )( msg[ CurrByte ] & 127 );
                }
                else
                {
                  cout << "Invalid message: Symbol longer than 4 chars" << endl;
                }
              }
              ++CurrByte; // Increment for next field
              break;
            }
          }
        }
        break;

      case 16:  // Check Quantity (allow for variable length)
        if( msg[ 0 ] & j )
        {
          bool decode_complete = false;
          m_Trade.Qty = 0;

          do
          {
            if( msg[ CurrByte ] & 128 )
            {
              m_Trade.Qty = ( msg[ CurrByte ] & 127 ) + m_Trade.Qty;
              decode_complete = true;
            }
            else
            {
              m_Trade.Qty = ( msg[ CurrByte ] & 127 );
              m_Trade.Qty = m_Trade.Qty << 7; // bit shift
              ++CurrByte;
            }
          } while( decode_complete == false );
        }
        break;

      case 8:  // Check Price (allow for variable length mantissa)
        if( msg[ 0 ] & j )
        {
```

```
          bool decode_complete = false;
          m_Trade.Price = 0;

          ++CurrByte;
          if( msg[ CurrByte ] & 64 )  // Decode exponent
          {
            exp = 0 - (( ~msg[ CurrByte ] & 127) + 1 ); // Negative value
          }
          else
          {
            exp = ( msg[ CurrByte ] & 127 ); // Positive value
          }
          ++CurrByte;
          do   // Decode mantissa
          {
            if( msg[ CurrByte ] & 128 )
            {
              m_Trade.Price = ( msg[ CurrByte ] & 127) + m_Trade.Price;
              decode_complete = true;
            }
            else
            {
              m_Trade.Price = ( msg[ CurrByte ] & 127 ) + m_Trade.Price;
              m_Trade.Price = m_Trade.Price * 128; //bit shift
              ++CurrByte;
            }
          } while( decode_complete == false );

          m_Trade.Price = ( m_Trade.Price ) * ( pow( 10, ( double ) exp ) );
          // Calculate final price
        }
        break;

      case 4:  // Check for other fields
          if( msg[ 0 ] & j )
          {
            cout << "Other field is present." << endl;
          }
          break;
      case 2:  // Check for other fields
          if( msg[ 0 ] & j )
          {
            cout << "Other field is present." << endl;
          }
          break;
    }
  }
  cout << "B/S:" << "\t" << m_Trade.BS << endl;
  cout << "SYMBOL:" << "\t" << m_Trade.Symbol <<  endl;
  cout << "QTY:" << "\t" << m_Trade.Qty << endl;
  cout << "PRICE:" << "\t" << m_Trade.Price << endl;
  return 0;
}
```