SD4 Assignment 2: Custom Binary Files

# Summary

* BufferWriter class, which affords appending various primitives into a growing binary byte-buffer
* BufferParser class, which affords scanning & parsing primitives from a fixed / existing binary buffer
* Save convex scene to correct format-conformant Guildhall Convex Scene (.GHCS) binary file
* Load convex scene from correct format-conformant Guildhall Convex Scene (.GHCS) binary file
* Correct handling of .GHCS file data (headers, ToC, data chunks), including unsupported chunk types
* Correct handling of little & big endian buffer data

# Requirements (maximum possible score is 110/100):

* (26) BufferWriter class
  + (4) Constructs from non-const reference (or pointer) to dynamic byte-array, e.g. std::vector<byte\_t>, on which it operates and pushes data into
  + (10) The ability to append each of the following primitive types onto the end of the buffer:
    - Byte, char, ushort, short, uint, int, uint64, int64, float, double
  + (4) The ability to append strings to the end of the buffer, using at least two different methods:
    - Zero-terminated (string begins immediately, and its end is marked with a zero-byte)
    - Length-preceded (uint32 string length, then exactly that many characters follow – no terminator)
  + (4) The ability to append some number of engine “semi-primitive” types (1 point each), for example:
    - Vec2/3/4, IntVec2/3/4, Rgb8, Rgba8, AABB2/3, OBB2/3, Plane2/3, VertexPCU, VertexLit, etc.
  + (2) The ability to overwrite a uint32 anywhere in the existing buffer (e.g. update ToC location)
  + (2) The ability to write in little- or big-endian format, with the machine’s native endian as default
* (30) BufferParser class
  + (4) Constructs from any data pointer + size (useful for piecemeal-parsing binary data structures)
  + (2) Constructs from a const reference (or pointer) to a dynamic byte-array, e.g. std::vector<byte\_t>
  + (10) The ability to parse (and advance) each of these primitive types from the “current” read position:
    - Byte, char, ushort, short, uint, int, uint64, int64, float, double
  + (4) The ability to parse (and advance) both string representations from the “current” read position:
    - Zero-terminated (string begins immediately, and its end is marked with a zero-byte)
    - Length-preceded (uint32 string length, then exactly that many characters follow – no terminator)
  + (4) The ability to parse (and advance) “semi-primitive” types from the read position:
    - Vec2/3/4, IntVec2/3/4, Rgba8, AABB2/3, OBB2/3, Plane2/3, VertexPCU, VertexLit, etc.
  + (2) The ability to jump the current read position to any given offset within the buffer
  + (2) The ability to read in little- or big-endian format, with the machine’s native endian as default
  + (2) Safety checks to prevent/error/throw if attempted to read beyond the end of the buffer
* (27) Save Convex Scene to .GHCS
  + (20) The ability to write out a valid, intact convex object scene to a custom binary file format (.GHCS), conformant with specifications described in the “**GHCS-C29 File Format**” Google Doc (linked in Canvas)
  + (5) Support for writing out at least one additional, or “non-canonical”, data chunk type (see specification)
    - Note: I can only write the bounding discs chunk, I didn’t get a chance to write out my quad tree
  + (2) Support for preserving (writing back out) unrecognized chunk types intact **if** the scene was previously loaded and unmodified (and discarding unsupported chunk types if the scene was modified)
* (27) Load Convex Scene from .GHCS
  + (20) The ability to read in a valid, intact convex object scene from a custom binary file format (.GHCS), conformant with the specifications document linked above.
  + (4) Support for reading in at least one “non-canonical” data chunk type
    - Note: I can only read the bounding discs chunk, I didn’t get a chance to read quad tree chunks
  + (3) Support for reading in and processing the SceneInfo chunk data, and scaling your scene accordingly
    - Note: When reading in a scene that is a different size, the objects are drawn correctly and have the correct bounding disc but the quad tree will always be the initial quad tree size and the quad checks won’t work correctly for scenes with larger sizes than my default

# Submission

* Use your own Guildhall “SD” C++ engine – maintained, buddy-tested, and submitted via Perforce.
  + A committed changelist in P4 with submission comment “SD4-A2: COMPLETE”.
  + Be sure to include an updated ReadMe, a current Release-built .EXE, and all required code & data files.
  + *See notes from Professor Service’s DFS2 class regarding maintenance of your code across P4 / GitHub.*
* Submit a **.zip file** to Canvas named **C29\_SD4\_*A2*\_*p4username*.zip** (*for example:* ***C29\_SD4\_A2\_beiserloh.zip***) which contains:
  + A very short (informal) voice-narrated **video** demonstrating the full functionality of your project;
  + A copy of **this document**, with completed items highlighted cyan, omitted items highlighted red, and partially completed items highlighted yellow (with inserted bullets-text underneath explaining).