Zip Code Group Project 3.0
Final Design Document

Date: 11/19/2023
Team 1 Members: Tristan Adams, Kent Biernath, Andrew Clayton, Emma Hoffmann

Project Overview
The primary objective of the Zip Code Group Project 3.0 is to advance the development of the C++ application founded in Group Projects 2.0 and 1.0. The following tasks and developments are included:

- Transform XLSX file into CSV (comma separate) format, then convert the CSV file to a file structure format where the records are length indicated.
- Generate blocked sequence set file from the data file created in Group Project 2.0
    - Capacity size of blocks can be specified (ex: 75% full).
- Use buffer classes for sequential processing of blocked sequence set files.
    - The block buffer unpacks records from a block into a record buffer.
        - This includes metadata for the size of a block, and the capacity that blocks are filled to. This ended up being handled by the header record buffer.
    - The record buffer unpacks fields into a record object.
- Create two dump methods to aggregate Zip Codes into blocks with RBN links, listing blocks by physical or logical order with initially identical outputs.
- Create a simple index file containing ordered pairs of keys (the highest key in each block) and block numbers, as well as a readable dump of the simple index.
- Develop system to generate, write, and read a simple primary key index in RAM for displaying Zip Code data for specified Zip Codes via command line.
- Enhance the search functionality to include options for both valid and invalid Zip Code searches, and to handle scenarios where a Zip Code record is not in the file.
- Enable the addition and deletion of records, including scenarios of block splits, merges, redistributions, logging, dumps, and index modifications. (This was never reached in time.)

Components
1. CSV Conversion and Data File Structure
    a. Manually convert the XLXS files to CSV files using the “Save As” option in Microsoft Excel.
    b. Convert CSV file to file structure format, beginning with a header record, with fields comma separated and records length indicated.
        i. Purpose: Modifies CSV files to a structured format containing length-indicated header records, enabling more functional data access.
            1. Reads input CSV file line by line
            2. Create a header record

Commented [HR1]: Most of the current design doc text is from Project 1 design doc. We should use this as reference and make appropriate adjustments and changes according to Project 2 requirements.

Please view Kent's comment.

Highlight changed areas that now comply with P2 requirements and add a comment saying "adjusted"

To ensure accuracy, we should all be reviewing each others work.

Commented [CM2]: I'm just trying to get more details in right now, feel free to readjust as needed for correctness or to make it easier to understand

Commented [CM3R2]: or if having several sub bullet points is messy that's something I can avoid too

Commented [BA4]: The metadata for the size of the blocks will be in the file data header, not in the block metadata. The block metadata contains the number of records in the block and links to the preceding and succeeding blocks.

3. Iterate through the header fields
4. Find the length of each header field
5. Prepend each record with a field containing the number of characters in the record before the length field is added, excluding the length field itself and the newline character between records.
6. Write to a new file using length-indicated records as each record is processed.

2. Header Record Architecture
    a. Purpose: Store the details about the data contained within the file including several properties about how to parse the records.
    b. Contains the following properties:
        i. File structure type
        ii. Version of the file structure in case the version changes later
        iii. Header record size in bytes
        iv. If fixed size, the count of bytes for each record size integer
        v. Size format type: ASCII size format
            1. Support for binary in future
        vi. Block size
        vii. Minimum Block Capacity
        viii. Primary key index file name
        ix. Primary key index file schema
        x. Record count
        xi. Block count
        xii. Count of fields per record
        xiii. Primary key indication (ordinally)
        xiv. For each field:
            1. Name or ID
            2. Data type (read/write)
        xv. RBN link to the block avail-list
        xvi. RBN link to the active sequence set list
        xvii. Any other fields that could be useful
            1. Data field indicating everything beyond is data records

    c. The header record will be across multiple lines at the beginning of the file for enhanced readability.

3. Block Architecture
    a. Purpose:
    b. Each active block contains the following properties:
        i. Count of records (> 0)
        ii. Links to preceding and succeeding active blocks
        iii. Set of records ordered by key

c. Each avail list contains the following properties:
    i. Count of records (== 0)
    ii. Link to succeeding avail block
    iii. All other bytes are overwritten with blanks

4. Blocked Sequence Set File Generation and Handling
    a. File Generation:
        i. Using data file generated in Project 2.0 to create the blocked sequence set file.
            1. Command line options to specify:
                a. The name of the blocked sequence set data file.
                b. All information necessary for the header file.
    b. Block Size Consistency:
        i. Default size is detailed in the Header Record Architecture section.
        ii. Block contents:
            1. A complete set of records - some blocks may contain different counts of records.
            2. Meta data architecture is detailed in the Block Architecture section.
        iii. Handling unused or deleted blocks:
            1. Convert unused or deleted blocks are avail list blocks as outlined in Folk 6.2.2 & 10.1-10.3
5. Buffer Class Functionality and File Processing
    a. Block Buffer Class:
        i. Handles block sequence set files, unpacking records from each block into a record buffer.
    b. Record Buffer Class:
        i. Unpacks individual fields the record buffer into record objects.
    c. Data File Header Record Buffer Class
        i. Modified to read and write the header record of the blocked sequence set data file.
        ii. Added the ability to count/update its own byte size value
6. Dump Methods and Index File Creation
    a. Blocked Sequence Set Dump Methods:
        i. Purpose: To visually aggregate and display Zip Codes within blocks, aiding with readability.
        ii. Two methods:
            1. Physical Ordering: Lists blocks in the order they physically appear in the file.
            2. Logical Ordering: Lists blocks based on a logical sequence, independent of their physical arrangement.
        iii. Consistent output initially, with different post modifications such as non-appending avail block usage.
    b. Index File Creation:

i. Simple Index File: Develops an index file containing ordered pairs of keys (highest key in each block) and block numbers, per Folk Figure 10.3 guidelines.
ii. Readable Dump: Ensures the creation of a dump from the simple index file that is easily interpretable.
iii. Functionality: To efficiently locate and reference specific blocks within the sequence set file.

c. Primary Key Index Generation
    i. RAM Processing: Generates a primary key index in RAM, writes it as a file, and then reads it back into RAM for optional use.
        1. Create the primary key index in RAM
            a. Read file
            b. Set zip code as key with records as values
        2. Write index to file
            a. Open file for writing
            b. Write index data into file
        3. Read index from file
            a. Open index file for reading
            b. Read index data and load into RAM
    ii. Usage: Aids in displaying Zip Code data for all Zip Codes listed via command line flags.
    iii. Structure: Index stores ordered pairs consisting of the highest key in each block and the corresponding Relative Block Number (RBN).

7. Search Functionality and Index Utilization
    a. Enhanced Search:
        i. Search with Flags: Search functionality allowing for search flags. Format: -Z12345 or -z12345, where the numbers are the ZIP code.
        ii. Flag Handling: Ability to handle multiple searches with multiple flags at once.
    b. Utilization of Primary Key Index for Searches:
        i. Record Location: Utilizes the primary key index for quick location of Zip Code records.
    c. Search Results Display and Handling:
        i. Record Display: Shows complete Zip Code record if found or if not found, a message indicating its absence is displayed.
    d. Integration with Blocked Sequence Set File:
        i. Interaction: Ensures the search function works efficiently with the blocked sequence set file.
        ii. Data Retrieval: Loads only necessary data into RAM for searches by locating the block to read in from the simple index
8. Application Program

a. Purpose: Uses the functionality of projects 1.0 and 2.0 with the additions of this project to read and write data stored in the block format. The zip code data will be stored and manipulated in this block format and the goal of the main program will be to create user functionality with the meta data through the command line. Things like record addition and deletion, zip code searching, and other ways to display data.
b. Accepts CSV or length-indicated files of ZIP code records with records sorted in any order and six fields per record in this order: ZIP Code, Place Name, State Code, County, Latitude, and Longitude.
c. Utilizes "ZipCodeBuffer" to read one record at a time.
d. Stores the records in a set and two maps:
    i. The set contains the state codes in alphabetical order and without duplicates.
    ii. One map stores the current coordinate extrema for the given state in a vector of doubles ordered easternmost longitude, westernmost longitude, northernmost latitude, and southernmost latitude.
    iii. The other map stores the string ZIP code associated with the coordinate extrema stored in the other map. It is a vector of strings sorted in the same way as the other map.
e. Stores the ZIP code record objects in a map for retrieval using the primary key index.
f. Sorts the records alphabetically by state codes.
g. Handles ZIP codes of variable lengths (not all codes in the data are 5 digits).

9. Record Management Test:
    a. Purpose: Evaluate the functionality of the blocked sequence set for Zip Code data and its integration with the simple index file.
    b. Search Test:
        i. Functionality: Create and execute a search test program that utilizes command line inputs for various search queries.
            1. Valid and Invalid Searches: Includes searches for several valid Zip Codes and at least one invalid Zip Code, testing the program's ability to identify and respond to each.
            2. Index Loading: Ensure program loads the simple primary key index file into a sorted container object in RAM without loading the entire blocked sequence set file.
    c. Record Addition and Deletion:
        i. Purpose: Assess application's ability to handle the addition and deletion of records within the blocked sequence set.
        ii. Record Addition:
            1. If a block is split or the index must be modified, log the event.
            2. Optionally, run the two dumps (split), or run a dump of the index.
        iii. Record Deletion:

Commented [BA5]: Try to ensure that most places that mention CSV also mention the length-indicated files that we need to handle.

1. If two blocks are merged, or participants of a redistribution, log the event.
2. Optionally, run the two dumps (merge or redistribution), or run a dump of the index.

iv. This section was not reached during the project by the deadline, so we did not get a chance to implement this strategy.

10. Documentation
    b. Extensive code comments and appropriate Doxygen tags.
        i. Includes class descriptions, method details, and parameter descriptions.
    c. Doxygen configuration file ("Doxyfile") for PDF documentation.
    d. User guide explaining how to run the program and how to interpret the results.
    e. Program test document.

Testing

The testing for Zip Code Group Project 3.0 is comprehensively documented in a separate test document, which details specific test cases and their expected outcomes. The test document provides a focused evaluation of the project's performance, accuracy, and efficiency, building upon the foundations laid in the previous projects.

Special Considerations

In addition to the main components and testing, special cases and considerations have been identified:

- Scalability: The system architecture supports growth, ensuring smooth performance even with significant increases in data volume.

Summary

The design document provides an overview of the core elements and considerations in Zip Code Group Project 3.0 while incorporating essential functionalities from Projects 1.0 and 2.0.