Names: Kevin Gutierrez, Kiran Kadariya, Sagar N, Abdul

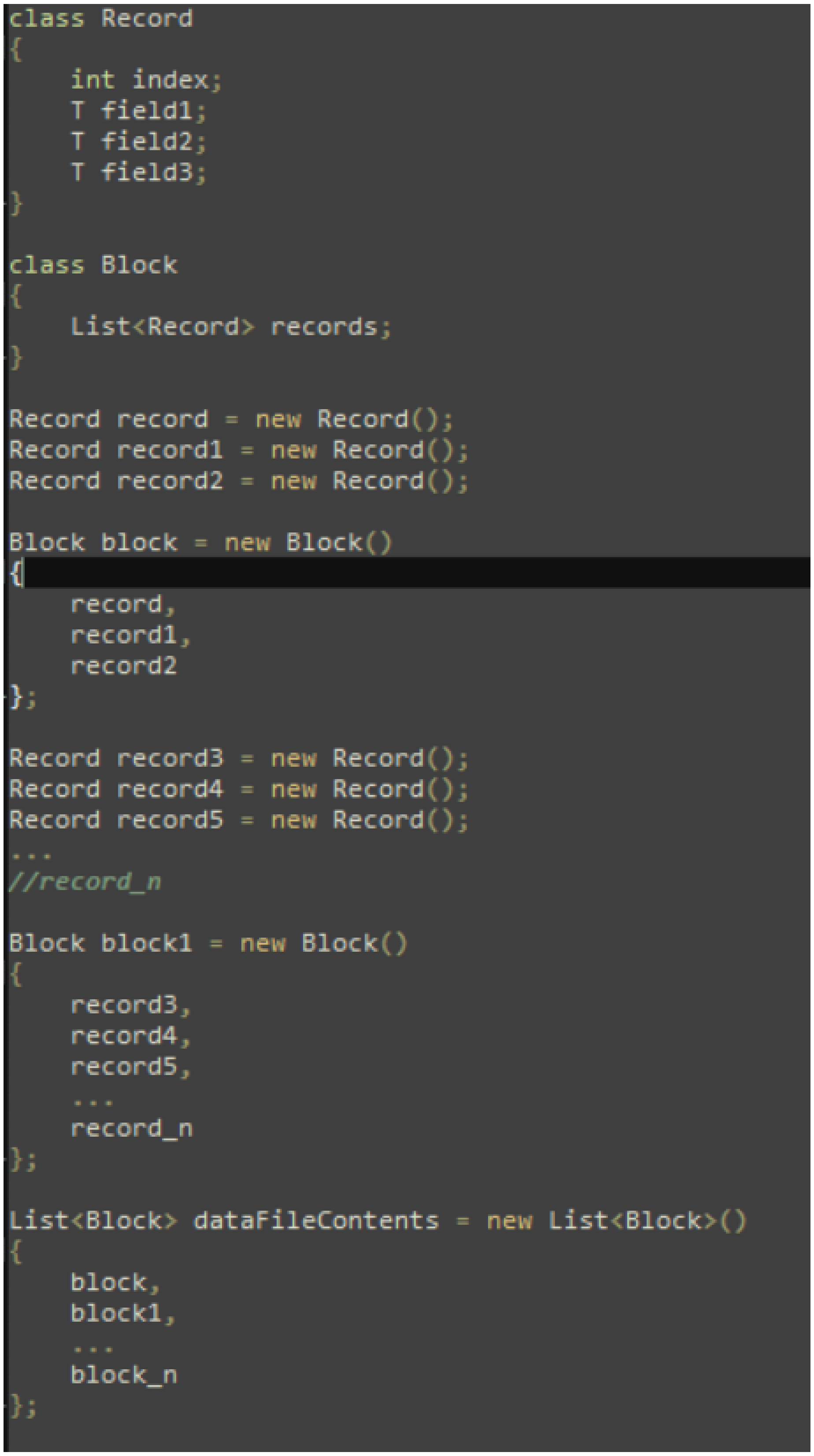
CSCI 331-54

Professor: Andrew Anda

Date: 4/8/25

Zip Code Group Project 2.0 Design Document

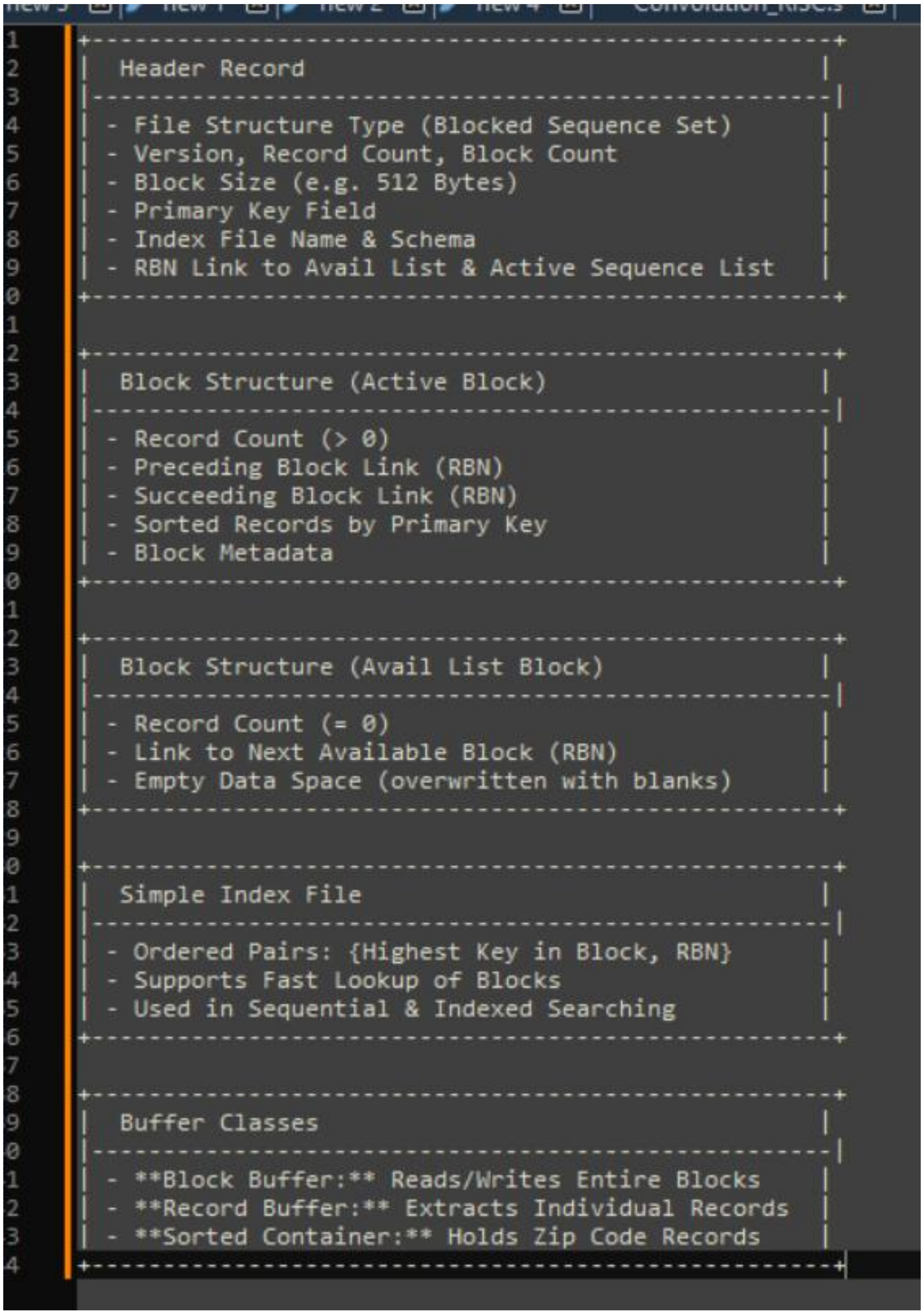
1. The project 2.0 codebase will be used as a basis for project 3.0. Using 2.0 as a basis, a blocked sequence set file will be defined by parsing the data file from 2.0, such that each block of the sequence set will be of equal size.
2. The sequence set file requires some config – this will be supplied using command line args.
3. The sequence set in (1) will be processed using buffer classes from 2.0
4. Two new classes will be created - BlockBuffer and RecordBuffer
   Notice! The following will represent the data structures – I wrote it in C# as that's what I find to be read the easiest in place of pseudocode; of course the related data structures in c++ can be deduced from the following relational data units (classes):



```
class Record
{
    int index;
    T field1;
    T field2;
    T field3;
}

class Block
{
    List<Record> records;
}

Record record = new Record();
Record record1 = new Record();
Record record2 = new Record();

Block block = new Block()
{
    record,
    record1,
    record2
};

Record record3 = new Record();
Record record4 = new Record();
Record record5 = new Record();
...
//record_n

Block block1 = new Block()
{
    record3,
    record4,
    record5,
    ...
    record_n
};

List<Block> dataFileContents = new List<Block>()
{
    block,
    block1,
    ...
    block_n
};
```

5. Using the blocked sequence set file from (1), we'll use the block sequenced file from (1) to create a data file like in project 1.
6. Then we'll create a dump method that will aggregate the zip codes, such that the dump will look like a linked list, except instead of hex memory addresses, it'll use a relative block number (essentially an incrementing primary key integer index – **for blocks**). The contents between each key will be the key for each **record,** and since each record contains k fields but exactly one key, there is a one to one relationship between keys and records!
7. A simple index file will be created containing ordered pairs of keys – the highest of each block and block numbers (RBN's)
8. Using the simple index, we'll create a readable dump.
9. Simple primary keys will be used to display zip code data from all zip codes listed in the command line, in the structure of a key of highest key in block and a value of relative block number. We will add functionality to search based on these files.

Data structures:



```
+-----------------------------------------------------+
|  Header Record                                      |
|-----------------------------------------------------|
| - File Structure Type (Blocked Sequence Set)        |
| - Version, Record Count, Block Count                |
| - Block Size (e.g. 512 Bytes)                       |
| - Primary Key Field                                 |
| - Index File Name & Schema                          |
| - RBN Link to Avail List & Active Sequence List     |
+-----------------------------------------------------+

+-----------------------------------------------------+
|  Block Structure (Active Block)                     |
|-----------------------------------------------------|
| - Record Count (> 0)                                |
| - Preceding Block Link (RBN)                        |
| - Succeeding Block Link (RBN)                       |
| - Sorted Records by Primary Key                     |
| - Block Metadata                                    |
+-----------------------------------------------------+

+-----------------------------------------------------+
|  Block Structure (Avail List Block)                 |
|-----------------------------------------------------|
| - Record Count (= 0)                                |
| - Link to Next Available Block (RBN)                |
| - Empty Data Space (overwritten with blanks)        |
+-----------------------------------------------------+

+-----------------------------------------------------+
|  Simple Index File                                  |
|-----------------------------------------------------|
| - Ordered Pairs: {Highest Key in Block, RBN}        |
| - Supports Fast Lookup of Blocks                    |
| - Used in Sequential & Indexed Searching            |
+-----------------------------------------------------+

+-----------------------------------------------------+
|  Buffer Classes                                     |
|-----------------------------------------------------|
| - **Block Buffer:** Reads/Writes Entire Blocks      |
| - **Record Buffer:** Extracts Individual Records    |
| - **Sorted Container:** Holds Zip Code Records      |
+-----------------------------------------------------+
```