1 Assignment Goals

1. Experience feature addition during the iterative design process.
2. Learn how to use the \texttt{vector} standard library template.
3. Learn how to use the \texttt{stringstream} standard library class.
4. Learn how to create and initialize circularly dependent classes.
5. Learn why circular \texttt{#include} do not work and how to deal with the include files for circularly dependent classes.
6. Learn how to use a public \texttt{enum} defined inside a class.
7. Learn a trick for finding the number of constants defined in an \texttt{enum}.
8. Learn how to use a static class constant.
9. Gain further experience in using private member functions to modularize overly-long public functions.

2 Problem

2.1 This Week’s Assignment

In problem sets 3 and 4, you implemented a Sudoku board that aggregates an array of \texttt{BSquare}, where a \texttt{BSquare} augments the \texttt{Square} class of problem set 2. In this assignment, you will extend the board design with data structures for representing Sudoku clusters.

Before you begin this assignment, you should fix any remaining errors in PS4 so that you’re starting from a correct working code base. If you are still having trouble getting your code to work after reading the feedback from the graders, then please write to \texttt{cs427help@cs.yale.edu} for help. You will not be able to do this assignment successfully if the code you start from still has bugs.

2.2 Sudoku Clusters

A \textit{cluster} is a set of squares that must be filled in with distinct marks. For this assignment, you may restrict your code to handling the three cluster types of classical Sudoku: rows, columns, and boxes. We will see additional cluster types in some variants of Sudoku.

A $9 \times 9$ Sudoku board has 27 clusters – 9 rows, 9 columns, and 9 boxes, where a box is a $3 \times 3$ block of squares. Because we permit boards smaller than $9 \times 9$, there may be fewer than 27 clusters, and not all clusters will be full size. For example, on an $8 \times 5$ board, there will be 2 clusters of size $3 \times 3$, 2 clusters of size $3 \times 2$, 1 cluster of size $2 \times 3$, and 1 cluster of size $2 \times 2$.

You will need to do three things to add clusters to your PS4 code:
1. You will need to define a new class `Cluster`. Each `Cluster` object will represent a set of Sudoku squares that must be filled in with distinct marks.

2. You will need to extend class `Board` to create all of the clusters in the board. `Board` should maintain a master list of all `Cluster` objects. It needs this so that it can print out all clusters in the board and can delete the clusters when the board is deleted.

3. You will need to extend class `BSquare` with a list of back pointers to the `Cluster` objects of which it is a member. Each time a new `Cluster` is created, its constructor will register the new cluster with each `BSquare` it contains. It does this by calling a new `BSquare` function `addCluster(Cluster* clup)` whose job is to add its cluster pointer argument to its list of back pointers.

   This kind of registration process where a newly-created object (the cluster) announces itself to interested clients (the squares) is sometimes called a callback mechanism.

2.3 Specifications for class `Cluster`

A `Cluster` should have data members to store the type of cluster (whether row, column, or box), a cluster ID used to distinguish among clusters of the same time, the number of `BSquare`s in the cluster, and a C-style array `bsq[]` of pointers to the `BSquare`s that belong to the cluster.

You should define a global int constant `maxCluSize = maxMark-minMark+1`, where `maxMark` and `minMark` are the global constants defined in `Square.hpp`. This is the maximum number of squares permitted in any cluster. (It would not make sense for a cluster to have more squares than this since then it would be impossible to fill in the squares with distinct marks.) Hence, `bsq[]` can have fixed size `maxCluSize` and can be composed in `Cluster` rather than aggregated.

Rows, columns, and boxes are each numbered sequentially, starting with 1. Rows are numbered from top to bottom, columns from left to right, and boxes in row-major order.

Here is the public interface for class `Cluster`.

- `ClusterType` is an `enum` type with constants ROW, COL, BOX, numClusterType, in that order. By default, `enum` will number them starting with 0, so `numClusterType`, the last `enum` constant, will be numbered 3, which is also the number of different cluster types in this assignment. `ClusterType` should be defined inside the class.

- `cluPrintName` is a static const char* array. It should be initialized (in `Cluster.cpp`) to the print names of the cluster types, that is, "Row", "Col", "Box".

- `Cluster(ClusterType typ, int id, int size, BSquare* bsqArg[])` creates a `Cluster` object with the given type and id. It should initialize its `bsq[]` array from the pointers in its fourth argument, `bsqArg[]`. You can’t to any sort of clever array or pointer assignment; rather, the `BSquare` pointers need to be copied one at a time from `bsqArg[]` to `bsq[]`. For each such pointer `p`, the constructor should register the cluster with `*p` by calling the function `p->addCluster(this)`. The third argument, `size`, is the number of pointers in `bsqArg[]`, which may be less than `maxCluSize`.

- `ostream& print(ostream& out) const` prints out a line describing the cluster, e.g.,

```plaintext
Box 6: (4, 7) (4, 8) (4, 9) (5, 7) (5, 8) (5, 9) (6, 7) (6, 8) (6, 9)
```
• `string stringName() const` creates and returns a string that uniquely identifies the cluster, e.g., Box 6. It builds this string from the typ and id fields passed to the `Cluster` constructor used to create the cluster.

In addition, the output operator `<<` should be extended as usual to print `Cluster` objects.

### 2.4 Extensions to BSquare

You should augment class BSquare as follows:

- Add a private data member `vector<Cluster*> clu` which will store a list of pointers to the clusters of which this BSquare object is a member.

- Add a public method `void addCluster(Cluster* clup)` that will append its argument `clup` to the cluster vector `clu`.

- Add a public method `string stringName() const` that creates and returns a string that uniquely identifies this BSquare, e.g., (4, 7) uniquely identifies the BSquare in row 4, column 7 of the Sudoku board.

- Modify the `print()` function to print the names of the clusters of which this BSquare is a member in addition to all of the information that it used to print. The format of the output should follow this example:

  BSquare (2, 4): in Row 2, Col 4, Box 2; Mark=4 (immutable); possibilities: 123_56789

  Also, refactor `print()` to make use of the new `stringName()` function.

### 2.5 Extensions to Board

You should augment `Board.hpp` with a private data member `vector<Cluster*> clu` which will store a list of all of the clusters on the board. The `Board(...)` constructor needs to also call three new private functions

- `void makeRowClusters();`

- `void makeColClusters();`

- `void makeBoxClusters();`

to create the clusters. The job of each of these functions is to create all of the clusters of its type. For example, to create box 6, it is necessary to compute the 9 squares that belong in that box, put them in a temporary array, create a new cluster, and put the pointer to it returned by `new` into the vector `clu`. The board destructor should of course be modified accordingly to delete the clusters that the constructor created.

You should also modify `Board::print()` to print out all of the clusters following the printout of all of the BSquares. You may need to make some minor formatting changes to match the sample inputs and outputs, which you will find in the assignment directory `/c/cs427/assignments/ps5/`.

### 3 Deliverables

Submit all of the files needed to build your project, whether or not they have changed from previous submissions. Be sure to submit your `Makefile` and `README` files. Remember to specify problem set 5 to the `submit` script.
4 Grading Rubric

Your assignment will be graded according to the scale given in Figure 1 (see below).

<table>
<thead>
<tr>
<th>#</th>
<th>Pts.</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1</td>
<td>A well-formed Makefile or makefile is submitted that specifies compiler options \texttt{-O1 \ -g \ -Wall}.</td>
</tr>
<tr>
<td>2.</td>
<td>1</td>
<td>Running make successfully compiles and links the project as specified in the assignment and results in an executable file \texttt{clusters}.</td>
</tr>
<tr>
<td>3.</td>
<td>1</td>
<td>\texttt{valgrind} detects no memory leaks.</td>
</tr>
<tr>
<td>4.</td>
<td>1</td>
<td>Class \texttt{Cluster} uses a C-style array of fixed dimension to store a list of pointers to the squares belonging to the cluster.</td>
</tr>
<tr>
<td>5.</td>
<td>1</td>
<td>Class \texttt{Cluster} defines an enum type with enum constants for identifying the cluster type.</td>
</tr>
<tr>
<td>6.</td>
<td>1</td>
<td>Class \texttt{Cluster} defines a static \texttt{char*} constant array whose elements contain the string constants corresponding to the enum constant names.</td>
</tr>
<tr>
<td>7.</td>
<td>1</td>
<td>The \texttt{Cluster} constructor performs a callback to register itself with the \texttt{BSquares} that belong to it.</td>
</tr>
<tr>
<td>8.</td>
<td>2</td>
<td>Classes \texttt{BSquare} and \texttt{Board} each use a \texttt{vector} of \texttt{Cluster*} to store a list of clusters.</td>
</tr>
<tr>
<td>9.</td>
<td>2</td>
<td>Class \texttt{BSquare} uses a forward declaration to break the circular dependency with \texttt{class Cluster}.</td>
</tr>
<tr>
<td>10.</td>
<td>1</td>
<td>The \texttt{Board} constructor uses private functions \texttt{makeRowClusters()}, \texttt{makeColClusters()}, and \texttt{makeBoxClusters()} to build the clusters.</td>
</tr>
<tr>
<td>11.</td>
<td>1</td>
<td>All data members are private.</td>
</tr>
<tr>
<td>12.</td>
<td>5</td>
<td>Running \texttt{clusters} produces output that matches the furnished output files.</td>
</tr>
<tr>
<td>13.</td>
<td>2</td>
<td>All relevant standards from previous assignments are followed regarding submission, identification of authorship on all files, commenting each function definition, and so forth.</td>
</tr>
</tbody>
</table>

20 Total points.

Figure 1: Grading rubric.