```
1
2  ////////////////////////////////////////////////////////////////////////////
3  //
4  // Author:           David Hawkins
5  // Email:            david.james@hawkinsonline.us
6  // Label:            P01
7  // Title:            Assignment 4 - Resizing the Stack
8  // Course:           3013
9  // Semester:         Spring 2020
10 //
11 // Description:
12 //       This program uses a class to implement an array-based stack. The stack
13 //       class is written in a way that allows the size of the stack to change
14 //       when it reaches capacity or when it becomes half-full. However, just
15 //       like a normal stack, it still pushes, pops, and checks if empty/full.
16 //       The program reads in from a file of large positive numbers, and if a
17 //       value read-in is even, it is pushed onto the stack; if the value is
18 //       odd, the program pops one of the even numbers off of the stack.
19 //
20 // Usage:
21 //       When you run the program, you will be prompted with message telling you
22 //       to write in the name of the file. Write the name of the file and press
23 //       enter, and the program will begin reading from the file you selected.
24 //       After the program has finished reading in values and altering the
25 //       stack, a prompt will appear that reads "Press any key to continue..."
26 //       Simply press a key and the program will stop running.
27 //
28 // Files:
29 //       main.cpp        : Driver program
30 //       nums.dat        : Input file
31 //
32 ////////////////////////////////////////////////////////////////////////////
33
34 #include <iostream>
35 #include <fstream>
36
37 using namespace std;
38
39 /**
40  * ArrayStack
41  *
42  * Description:
43  *       An array-based stack that pushes and pops values onto itself. It is
44  *       implemented in such a way that allows it to resize itself if it
45  *       reaches capacity or becomes half-full.
46  *
47  * Public Methods:
48  *                       ArrayStack()
49  *                       ArrayStack(int s)
50  *       bool            Empty()
51  *       bool            Full()
52  *       int             Peek()
53  *       int             Pop()
54  *       void            Print()
55  *       void            Push(int x)
56  *       void            ContainerGrow()
57  *       void            ContainerShrink()
58  *       void            CheckResize()
59  *
60  * Usage:
61  *       Used to hold values read-in from a file of 100000 numbers. When a value
62  *       is found to be even, it would get pushed onto the stack. When a value
63  *       is found odd, an item is popped off of the stack. Each time a value is
64  *       popped or pushed, the CheckResize method checks to see if the size of
65  *       the stack needs to be modified by checking to see if it's full or if
66  *       the ratio between (top + 1) and size is less than one-half.
67  *
68  */
69 class ArrayStack {
```

```
70  private:
71      int *A;           // pointer to array of int's
72      int size;         // current max stack size
73      int top;          // top of stack
74
75  public:
76      /**
77       * ArrayStack
78       *
79       * Description:
80       *      Default Constructor. Sets size to 10, dynamically allocates the
81       *      array, and sets top equal to -1.
82       *
83       * Params:
84       *     - None
85       *
86       * Returns:
87       *     - NULL
88       */
89      ArrayStack() {
90          size = 10;
91          A = new int[size];
92          top = -1;
93      }
94
95      /**
96       * ArrayStack
97       *
98       * Description:
99       *      Parameterized Constructor. Sets size to the value passed to it,
100      *      dynamically allocates the array, and sets top equal to -1.
101      *
102      * Params:
103      *     - None
104      *
105      * Returns:
106      *     - NULL
107      */
108     ArrayStack(int s) {
109         size = s;
110         A = new int[s];
111         top = -1;
112     }
113
114     /**
115      * Public bool: Empty
116      *
117      * Description:
118      *      Checks to see if the stack is empty or not.
119      *
120      * Params:
121      *      - NULL
122      *
123      * Returns:
124      *      - [bool] true = empty
125      */
126     bool Empty() {
127         return (top <= -1);
128     }
129
130     /**
131      * Public bool: Full
132      *
133      * Description:
134      *      Stack full?
135      *
136      * Params:
137      *      NULL
138      *
```

```
139         * Returns:
140         *       [bool] true = full
141         */
142        bool Full() {
143            return (top >= size - 1);
144        }
145
146        /**
147         * Public int: Peek
148         *
149         * Description:
150         *       Returns top value without altering the stack.
151         *
152         * Params:
153         *       - NULL
154         *
155         * Returns:
156         *       - [int] top value if any
157         */
158        int Peek() {
159            if (!Empty()) {
160                return A[top];
161            }
162
163            return -99; // some sentinel value
164                        // not a good solution
165        }
166
167        /**
168         * Public int: Pop
169         *
170         * Description:
171         *       Checks to see if the stack's size needs to be altered, then returns
172         *       top value and removes it from stack.
173         *
174         * Params:
175         *       - NULL
176         *
177         * Returns:
178         *       - [int] top value if any
179         */
180        int Pop() {
181
182            CheckResize();  //Checks to see is size needs to be altered
183
184            if (!Empty()) {
185                return A[top--];
186            }
187
188            return -99;     // some sentinel value
189                            // not a good solution
190        }
191
192        /**
193         * Public void: Print
194         *
195         * Description:
196         *       Prints stack to standard out
197         *
198         * Params:
199         *       NULL
200         *
201         * Returns:
202         *       NULL
203         */
204        void Print() {
205            for (int i = 0; i <= top; i++) {
206                cout << A[i] << " ";
207            }
```

```
208             cout << endl;
209         }
210
211         /**
212          * Public int: getSize
213          *
214          * Description:
215          *      Returns size of the stack
216          *
217          * Params:
218          *      - NULL
219          *
220          * Returns:
221          *      - [int] : size of the stack
222          */
223         int getSize() {
224             return size;
225         }
226
227         /**
228          * Public bool: Push
229          *
230          * Description:
231          *      Checks to see if the stack's size needs to be altered, then adds an
232          *      item to top of stack.
233          *
234          * Params:
235          *      - [int] : item to be added
236          *
237          * Returns:
238          *      - [bool] : success = true
239          */
240         bool Push(int x) {
241
242             CheckResize();  //Checks to see is size needs to be altered
243
244             if (!Full()) {
245                 A[++top] = x;
246                 return true;
247             }
248
249             return false;
250
251         }
252
253         /**
254          * Public void: ContainerGrow
255          *
256          * Description:
257          *      Resizes the container for the stack by increasing its capacity
258          *      by 1.75.
259          *
260          * Params:
261          *      - NULL
262          *
263          * Returns:
264          *      - NULL
265          */
266         void ContainerGrow() {
267             int newSize = size * 1.75;          // size of orig increases by 1.75
268             int *B = new int[newSize];          // allocate new memory
269
270             for (int i = 0; i < size; i++) {    // copy values to new array
271                 B[i] = A[i];
272             }
273
274             delete[] A;                         // delete old array
275
276             size = newSize;                     // save new size
```

```
277
278         A = B;                                  // reset array pointer
279
280     }
281
282     /**
283       * Public void: ContainerShrink
284       *
285       * Description:
286       *      Resizes the container for the stack by halfing its capacity.
287       *
288       * Params:
289       *      - NULL
290       *
291       * Returns:
292       *      - NULL
293       */
294     void ContainerShrink() {
295         int newSize = size / 2;                 //size of original cut in half
296         int *B = new int[newSize];
297
298
299         for (int i = 0; i < (top + 1); i++) {   // copy values to new array
300             B[i] = A[i];
301         }
302
303         delete[] A;                             // delete old array
304
305         size = newSize;                         // save new size
306
307         A = B;                                  // reset array pointer
308     }
309
310     /**
311       * Public void: CheckResize
312       *
313       * Description:
314       *      Checks if the container needs to be resized
315       *
316       * Params:
317       *      NULL
318       *
319       * Returns:
320       *      NULL
321       */
322     void CheckResize() {
323         if (Full()) {
324             ContainerGrow();
325         }
326         else if (((top + 1) < size / 2) && size >= 10) {    //checks ratio AND if
327             ContainerShrink();                              //stack is empty
328         }
329     }
330
331 };
332
333 /*************************************************
334  *
335  * Function Prototypes
336  *
337  *************************************************/
338
339 void openFiles(ifstream & infile, ofstream & outfile);
340
341 void readValue(ifstream & infile, int &v);
342
343 bool evenOrOdd(int val);
344
345 void printResults(ArrayStack sta, int resized, int max, ofstream & outfile);
```

```
346
347 void closeFiles(ifstream& infile);
348
349 // MAIN DRIVER
350 // Simple Array Based Stack Usage:
351 int main() {
352
353     ifstream infile;                        //input file object
354     ofstream outfile;                       //output file object
355     openFiles(infile, outfile);
356
357     ArrayStack stack;                       //stack object
358     int v = 0;                              //holds read-in values
359     int resize = 0;                         //how many times stack resized
360     int changeSize = 0;                     //holds size before resizing
361
362     readValue(infile, v);                   //read in first value
363
364     int maxSize = stack.getSize();          //get first size sets to maxsize
365
366     while(!infile.eof())                    //loops until end of file
367     {
368
369         changeSize = stack.getSize();       //records size before push/pop
370
371         if (evenOrOdd(v)) {
372             stack.Push(v);
373         }
374
375         else {
376             stack.Pop();
377         }
378
379         if(changeSize != stack.getSize()){  //checks if size was changed
380             resize++;                       //increments # of resizes
381         }
382         if(stack.getSize() > maxSize){      //if size > maxSize, then size
383             maxSize = stack.getSize();      //is the new maxSize
384         }
385
386         readValue(infile, v);               //read next value
387     }
388
389     printResults(stack, resize, maxSize, outfile);
390
391     closeFiles(infile);
392
393     system("pause");
394     return 0;
395
396 }
397
398 /**
399 * void: openFiles
400 *
401 * Description:
402 *       Opens the file to be read from
403 *
404 * Params:
405 *      - [ifstream] : stream object that reads from the file
406 *      - [ofstream] : stream object that writes to a file
407 *
408 * Returns:
409 *      - NULL
410 */
411 void openFiles(ifstream & infile, ofstream & outfile){
412     char inFileName[40];
413     char outFileName[40];
414     cout << "Enter the input file name: ";
```

```
415       cin >> inFileName; infile.open(inFileName);
416       cout << "Enter the output file name: ";
417       cin >> outFileName; outfile.open(outFileName);
418   }
419
420   /**
421   * void: readValue
422   *
423   * Description:
424   *       Reads in the next value from the file using the infile object
425   *
426   * Params:
427   *       - [ifstream] : used to read-in values
428   *       - [int&] : reference variable that holds read-in values
429   *
430   * Returns:
431   *       - NULL
432   */
433   void readValue(ifstream & infile, int &v){
434       infile >> v;
435   }
436
437   /**
438   * bool: evenOrOdd
439   *
440   * Description:
441   *       Checks to see if the value read in from the file is an even or odd #
442   *
443   * Params:
444   *       - [int] : value read in from the file
445   *
446   * Returns:
447   *       - [bool] : true = value was even
448   */
449   bool evenOrOdd(int val) {
450       if (val % 2 == 0) {
451           return true;       //if the value is even, function returns true
452       }
453       else {
454           return false;      //if the value if odd, function returns false
455       }
456   }
457
458   /**
459   * void: printResults
460   *
461   * Description:
462   *       Prints the results of maxSize, Ending Stack Size, and how many times
463   *       the stack was resized to the standard out.
464   *
465   * Params:
466   *       - [ArrayStack] : stack object. Get final size from this object.
467   *       - [int] : how many times the stack was resized
468   *       - [int] : the maximum size that the stack reached
469   *
470   * Returns:
471   *       - NULL
472   */
473   void printResults(ArrayStack sta, int resized, int max, ofstream & out) {
474       out << "#########################################################" << endl;
475       out << "\tAssignment 4 - Resizing the Stack" << endl;
476       out << "\tCMPS 3013" << endl;
477       out << "\tDavid Hawkins" << endl << endl;
478       out << "\tMax Stack Size: " << max << endl;
479       out << "\tEnd Stack Size: " << sta.getSize() << endl;
480       out << "\tStack Resized: " << resized << " times" << endl;
481       out << "#########################################################" << endl;
482   }
483
```

```
484  /**
485  * void: closeFiles
486  *
487  * Description:
488  *       Closes the files that were opened for reading.
489  *
490  * Params:
491  *      - [ifstream] : stream object used to read
492  *
493  * Returns:
494  *      - NULL
495  */
496  void closeFiles(ifstream& infile)
497  {
498      infile.close();
499  }
```