```cpp
/*****************************************************************************
 * PROGRAMMER : Ali Eshghi
 * STUDENT ID : 1112261
 * CLASS      : CS1B
 * SECTION    : MW 7:30pm
 * Assign #3  : Searching linked list
 * DUE DATE   : 6 November 2019
 *****************************************************************************/

#include "MyHeader.h"
#include "ClassHeader.h"


/*****************************************************************************
 * Methods for class StackList
 *****************************************************************************/

/*****************************************************************
 * StackList ();
 * Constructor; Initialize class attributes
 * Parameters: none
 * Return: none
 *****************************************************************/

StackList::StackList()
{
    stackCount = 0;
    head       = NULL;
}


/*****************************************************************
 * ~StackList ();
 * Destructor; does not perform any specific function
 * Parameters: none
 * Return: none
 *****************************************************************/
StackList::~StackList()
{
    DVDNode *DVDPtr;

    //Clear the list
    DVDPtr = head;
    while(DVDPtr != NULL)
    {
        head = head -> next;
        delete DVDPtr;

        DVDPtr = head;
    }
}


/*****************************************************************
 * void Push (DVDNode newDVD);
```

```
 56 *
 57 * Mutator; This method will add a DVD node to the list to the front
 58 *-------------------------------------------------------------------
 59 * Parameter: newDVD (DVDNode) //IN - node to be added to list
 60 *-------------------------------------------------------------------
 61 * Return: none
 62 ********************************************************************/
 63 void StackList::Push(DVDNode newDVD)
 64 {
 65     DVDNode *movPtr;    //In & Calc  - node to input data
 66
 67
 68     movPtr = head;
 69
 70     movPtr = new DVDNode; //adding new node for the list
 71     *movPtr = newDVD;     //constents copied
 72
 73     //add to the front
 74     movPtr -> next = head;
 75     head = movPtr;
 76
 77     //increment the stack count
 78     stackCount++;
 79
 80     movPtr = NULL;
 81
 82
 83 }
 84
 85 /********************************************************************
 86  * DVDNode Pop ();
 87  *
 88  * Mutator; This method will remove a DVD node from the front of the
 89  * list and return the DVDNode being removed
 90  *-------------------------------------------------------------------
 91  * Parameter: none
 92  *-------------------------------------------------------------------
 93  * Return: popDVD (DVDNode)
 94  ********************************************************************/
 95 DVDNode StackList::Pop()
 96 {
 97     DVDNode     searchPtr;
 98     DVDNode     *movPtr;
 99
100     movPtr = head;
101
102
103     if(IsEmpty())
104     {
105         cout << "The list is empty";
106
107         return searchPtr;
108     }
109
110     searchPtr = Peek();
```

```
111     movPtr = movPtr -> next;
112 
113     delete movPtr;
114 
115     movPtr = NULL;
116 
117     return searchPtr;
118 }
119 
120 /*****************************************************************
121  * bool IsEmpty () const;
122  *
123  * Accessor; This method will return the boolean value whether
124  * the list is empty or not empty
125  * ---------------------------------------------------------------
126  * Parameters: none
127  * ---------------------------------------------------------------
128  * Return: empty (bool)
129  *****************************************************************/
130 bool StackList::IsEmpty() const
131 {
132     bool empty;
133 
134     if(stackCount == 0)
135     {
136         empty = true;
137     }
138     else
139     {
140         empty = false;
141     }
142 
143     return empty;
144 }
145 
146 /*****************************************************************
147  * DVDNode Peek () const;
148  *
149  * Accessor; This method will return the DVD node of the first
150  * element on the list
151  * ---------------------------------------------------------------
152  * Parameters: none
153  * ---------------------------------------------------------------
154  * Return: returnDVD (DVDNode)
155  *****************************************************************/
156 DVDNode StackList::Peek() const
157 {
158     DVDNode DVDPtr;
159     DVDPtr.title = "EMPTY";
160 
161     if(IsEmpty())
162     {
163         DVDPtr = *head;
164     }
165 
```

```
166     return DVDPtr;
167
168 }
169
170 /*************************************************************
171  * int Size () const;
172  *
173  * Accessor; This method will return the size of the list
174  * -----------------------------------------------------------
175  * Parameters: none
176  * -----------------------------------------------------------
177  * Return: stackCount (int)
178  *************************************************************/
179 int StackList::Size() const
180 {
181     return stackCount;
182 }
183
184
185 /*************************************************************************************
186  * Methods for class MovieList
187  ************************************************************************************/
188
189 /*************************************************************
190  * MovieList ();
191  * Constructor; Initialize class attributes
192  * Parameters: none
193  * Return: none
194  *************************************************************/
195 MovieList::MovieList() {}
196
197 /*************************************************************
198  * ~MovieList ();
199  * Destructor; does not perform any specific function
200  * Parameters: none
201  * Return: none
202  *************************************************************/
203 MovieList::~MovieList() {}
204
205
206 /*************************************************************************
207  * void CreateList (string inputFileName);
208  *
209  * Mutator; This method will create a movie list using the input file
210  * data
211  *-----------------------------------------------------------------------
212  * Parameter: inputFileName (string) //IN - input file name
213  *-----------------------------------------------------------------------
214  * Return: none
215  ************************************************************************/
216 void MovieList::CreatList(string inFileName)
217 {
218     ifstreaminFile;
219
220     DVDNode     node;
```

```
221
222     inFile.open(inFileName);
223
224     while(inFile)
225     {
226         getline(inFile, node.title);
227         getline(inFile, node.leadActor);
228         getline(inFile, node.supActor);
229         getline(inFile, node.genre);
230         getline(inFile, node.altGenre);
231         inFile >> node.year;
232         inFile >> node.rating;
233         inFile.ignore(10000,'\n');
234         getline(inFile, node.synopsis);
235         inFile.ignore(10000,'\n');
236
237         StackList::Push(node);
238
239     }
240
241     inFile.close();
242 }
243
244
245 /*****************************************************************
246  * void OutputList (string outputFileName) const;
247  *
248  * Accessor; This method will output the list onto the output file
249  * ---------------------------------------------------------------
250  * Parameters: outputFileName (string) //IN - output file name
251  * ---------------------------------------------------------------
252  * Return: none
253  *****************************************************************/
254 void MovieList::OutputList(string outFileName) const
255 {
256
257     ofstreamoutFile;
258     int         movieCount;
259     string      plot;
260     DVDNode     *ptr;
261
262
263     outFile.open(outFileName);
264
265     movieCount = 0;
266     ptr = NULL;
267     ptr = head;
268
269     //printing the class header to the output file
270     PrintHeader(outFile, "OOP - DVD Movie List", 5, 'A');
271
272     while(ptr != NULL)
273     {
274         movieCount++;
275
```

```
276
277         outFile << endl;
278         outFile << left;
279         outFile << "*******************************************************";
280         outFile << "*******************************************************";
281         outFile << setw(18) << "MOVIE #: " << movieCount << "Title: "
282                 << ptr -> title
283                 << endl;
284         outFile << "-------------------------------------------------------";
285         outFile << "-------------------------------------------------------";
286         outFile << "-" << endl;
287         outFile << setw(18) << "Year: " << ptr -> year << "Rating: "
288           << ptr -> rating;
289         outFile << "-------------------------------------------------------";
290         outFile << "-------------------------------------------------------";
291         outFile << "-" << endl;
292         outFile << setw(57) << "Leading Actor:    " << ptr -> leadActor;
293         outFile << right;
294         outFile << "Genre 1: " << ptr -> genre << endl;
295         outFile << left;
296         outFile << setw(57) << "Supporting Actor: "
297           << ptr -> supActor;
298         outFile << right;
299         outFile << "Genre 2: " << ptr -> altGenre << endl;
300         outFile << "-------------------------------------------------------";
301         outFile << "-------------------------------------------------------";
302         outFile << "-" << endl;
303         outFile << "PLOT:" << endl;
304         plot = WordWrap(ptr -> synopsis);
305         outFile << plot << endl;
306         outFile << "*******************************************************";
307         outFile << "*******************************************************";
308
309         //next node on the list
310         ptr = ptr -> next;
311
312     }
313
314     outFile.close();
315
316 }
317
318
319 /***************************************************************
320  * string WorpWarp (string plot) const;
321  *
322  * Accessor; This method alter the string to wordwrap around a
323  * certain length of characters
324  * -------------------------------------------------------------
325  * Parameters: plot (string) //IN - synopsis
326  * -------------------------------------------------------------
327  * Return: returnStr (string)
328  **************************************************************/
329 string MovieList::WordWrap(string plot) const
330 {
```

```
331     const int MAX_SIZE = 75;
332     int       i;
333     int       size;
334     string    str;
335     string    line;
336     string    word;
337 
338     size = plot.length();
339 
340     str.clear();
341     word.clear();
342     line.clear();
343 
344     //plot
345     for(int i = 0; i <= size; i++)
346     {
347         if(plot[i] != ' ')
348         {
349             //concatenates chars until space is reached
350             word = word + plot[i];
351         }
352         else
353         {
354             if(line.length() + word.length() > MAX_SIZE)
355             {
356                 str = str + line + '\n';
357                 line.clear();
358             }
359         }
360 
361         line = line + word + " ";
362         word.clear();
363     }
364 
365     //output the last line
366     if(line != " ")
367     {
368         str = str + line + word + '\n';
369     }
370 
371     word.clear();
372     line.clear();
373 
374     return str;
375 
376 }
377 /*****************************************************************************
378  * FUNCTION PrintHeader
379  * ___________________________________________________________________________
380  *
381  * This function outputs a header including the lab name, lab number,
382  * the programmer's name, the class name, and the section time
383  * as a string by ostream
384  * ___________________________________________________________________________
385  * PRE-CONDITIONS:
```

```
386  * output - ostream variable to dynamically choose datatype
387  * of cout or ofstream
388  * labName - Name of the lab. The labName should be previously
389  * defined
390  * labNumber - Number of the lab. The labNumber should be
391  * previously defined
392  * labType - type of assignment
393  * 'A' - assignment
394  * 'L' - lab
395  *
396  * POST-CONDITIONS:
397  * outputs header as string
398  ********************************************************************/
399 void MovieList::PrintClassHeader(ostream&output,
400                                  string     labName,
401                                  int    labNumber,
402                                  char       labType) const
403 {
404     //Defining and initializing constant variables
405     const char PROGRAMMER[30] = "Ali Eshghi";
406     const char CLASS[5] = "CS1B";
407     const char SECTION[20] = "MW: 7:30p - 9:50p";
408 
409     // OUTPUT - Class Heading
410     output << left;
411     output << endl;
412     output << "**************************************************************";
413     output << "\n* PROGRAMMED BY : " << PROGRAMMER;
414     output << "\n* " << setw(14) << "CLASS" << ": " << CLASS;
415     output << "\n* " << setw(14) << "SECTION" << ": " << SECTION;
416     output << "\n* ";
417 
418     if (toupper(labType) == 'L')
419     {
420         output << "LAB #" << setw(8);
421     }
422     else
423     {
424         output << "ASSIGNMENT #" << setw(1);
425     }
426     output << labNumber << " : " << labName;
427     output << "\n**************************************************************";
428     output << "**\n\n";
429     output << right;
430 
431 }
432 
433 
434 
435 
436 
437 
```