```cpp
////
// Name: Jeff Calderon
// Section: CS132 33616
// Class Name: JCString
// Program Name: JCString Class Implementation
//
// Description: Constructor takes optional one arg "size"
// Member functions use comparisons between ASCII Values
// for operators <, >, ==, and their inverses.
// Other operators add JCStrings or char* arrays
// empty JCString are initialized with size number of '\0's
// this class can compare words and return weather arg word is
// greater or smaller by ASCII value
////

#include "JCStringV2.h"
#include <iostream>



// Static Variables
int JCString::currentCount = 0;
int JCString::createdCount = 0;


// initializer with basic values
// creates a char array with 7 memory
JCString::JCString(int size)
{
    jcStrInitialize(size, 0);//initialize new memory, size is cap and zero end index

    //keeping track of objects
    ++currentCount;
    ++createdCount;

```

```
36      this->str_num = createdCount;
37  }
38
39  // Destructor
40  JCString::~JCString()
41  {
42      // if initialized clear old stuff
43      if (this->str != nullptr)
44      {
45          // Be free memory
46          delete[] this->str;
47          this->str = nullptr;
48
49          //keeping track of objects
50          --currentCount;
51      }
52  }
53
54  // constructor for dumping arrays
55  JCString::JCString(const char* cstr)
56  {
57      //while loop counts chars and stores int
58      int numChars = 0;
59      for (numChars; cstr[numChars] != '\0'; ++numChars);
60
61      this->jcStrInitialize(numChars+2, numChars, cstr); // reserves memomory for chars
62
63      //keeping track of objects
64      ++createdCount;
65      ++currentCount;
66      this->str_num = createdCount;
67  }
68  // Copy constructor another JCString
69  JCString::JCString(const JCString& jcstr)
70  {
```

```
71      //constructor passes to jcstr init, whic handles the allocating  memory
72      this->jcStrInitialize(jcstr.cap, jcstr.end, jcstr.str); // reserves memomory
73
74      //keeping track of objects
75      ++currentCount;
76      ++createdCount;
77
78      this->str_num = createdCount;
79
80  }
81  // Initializing all values with null.
82  void JCString::charInitialize( const int capSize)
83  {
84      for (int i = 0; i < capSize; ++i)
85      {
86          this->str[i] = '\0';
87      }
88      // This makes sure we always have a null at the end
89      this->str[end] != '\0';
90
91
92  }
93
94  int JCString::length() const
95  {
96      return this->end;
97  }
98
99  int JCString::capacity() const
100 {
101     return this->cap;
102 }
103
104 //returns individual letter at index argument
105 char JCString::operator[](const int index) const
```

```
106 {
107
108     if (index >= 0 && index < this->end) {
109         return this->str[index];
110     }
111     else {
112         return '\0';
113     }
114 }
115
116 // reads in the word with the extractor operator ">>"
117  istream& operator>>(istream& inputStrm, JCString& jcstr)
118 {
119     char inputWord[ 100 ];
120     if (inputStrm >> inputWord)
121     {
122         int numChars = 0;
123         for (numChars= 0; inputWord[numChars] != '\0'; ++numChars);//Loop counts letters finds end
124
125         jcstr = inputWord; // Let copy assignment operator handle the resize
126     }
127     return inputStrm;
128 }
129
130 //write out char string to stream with insertion
131 ostream& operator<<(ostream& outputStrm, const JCString& jcstr)
132 {
133
134     outputStrm << jcstr.str;
135     return outputStrm;
136 }
137
138 bool JCString::operator<(const JCString& argStr) const
139 {
140     if (this->JCScompareTo(argStr) == -1)
```

```
141     {
142         return true;
143     }
144
145     return false;
146 }
147
148 JCString JCString::operator+(const JCString& rhsJCString) const
149 {
150
151     // Temp jcstring gets deleted after assignment operator copies to left hand side var
152     JCString comboString = appendCstr(this->str, rhsJCString.str);// Calls char dump constructor
153
154     return comboString;
155
156 }
157
158 JCString JCString::operator+(const char* rhsChars) const
159 {
160
161     // Temp jcstring gets deleted after assignment operator copies to left hand side var
162     JCString comboString = appendCstr(this->str, rhsChars);// concatenates then Calls char dump
        constructor
163
164     return comboString;
165
166 }
167 // Assignment to C_string
168 JCString& JCString::operator=(const char* strToCopy)
169 {
170     if (this->str != strToCopy)
171     {
172         if (this->str != nullptr)
173         {
174             // if initialized clear old stuff
```

```
175             delete[] this->str;
176         }
177
178         // count chars in string to copy
179         int numChars = 0;
180         for (numChars; strToCopy[numChars] != '\0'; ++numChars);
181
182         //assemble internal char array
183         jcStrInitialize(numChars+2, numChars, strToCopy);
184
185     }
186
187     return *this;
188 }
189 // Assignment to JCSting
190 JCString& JCString::operator=(const JCString& strToCopy)
191 {
192     if(this != &strToCopy)
193     {
194         if (this->str != nullptr)
195         {
196             // if initialized clear old stuff
197             delete[] this->str;
198         }
199
200         //assemble internal char array
201         jcStrInitialize(strToCopy.cap, strToCopy.end, strToCopy.str);
202
203     }
204
205     return *this;
206 }
207 // Assembles internal char array
208 // Cap size must be known in advance
209 // Old stuff deleted before calling jcString init
```

```
210 void JCString::jcStrInitialize(const int capSize, const int end, const char* inString/* =nullptr' */)
211 {
212     this->end = end;
213     this->cap = capSize;
214     this->str = new char[this->cap];
215 
216     //initializing all values with null.
217     this->charInitialize(this->cap); //this makes sure we always have a null at the end
218 
219     if(inString != nullptr)
220     {
221         //for loop checks that we don't go over the cap
222         for (int i = 0; i < this->cap && inString[i] != '\0'; ++i)
223         {
224             this->str[i] = inString[i];
225         }
226     }
227 
228 }
229 JCString& JCString::operator+=(const JCString& rhsJCString)
230 {
231 
232     JCString comboString = appendCstr(this->str, rhsJCString.str);// Calls char dump constructor
233     *this = comboString;
234 
235     return *this;
236 }
237 //defined in terms of the string compare function
238 bool JCString::operator>(const JCString& argStr) const
239 {
240     if (this->JCScompareTo(argStr) == 1)
241     {
242         return true;
243     }
244 
```

```
245     return false;
246 }
247
248 //defined in terms of the string compare function
249 bool JCString::operator==(const JCString& argStr) const
250 {
251     if (this->JCScompareTo(argStr) == 0)
252     {
253         return true;
254     }
255     return false;
256 }
257
258 // defined in terms of the == operator
259 bool JCString::operator!=(const JCString& argStr) const
260 {
261     return !(this->operator==(argStr));
262 }
263
264
265 // compares char for char, returns 1 if this-> string is larger ascii value than argument jcstring
266 // returns -1 if this-> string is lesser ascii value
267 // returns 0 if this-> string is same ascii value
268 int JCString::JCScompareTo(const JCString& angStr) const
269 {
270
271         int len = 0;
272         int count = 0;
273         int result = 0;
274
275         // dummie char stirngs
276         JCString thisString(this->str);
277
278         // lower case logic for when we implement that later
279         // uncomment when ready
```

```
280         // TODO: write a method which makes this->str lowercase
281         //JCString str2 = angStr.returnLower();
282         //str1.makeLower();
283
284         JCString str2(angStr.str);
285         // make sure we iter through to the shortest char string
286         if (thisString.length() < str2.length())
287         {
288             len = thisString.length();
289         }
290         else
291         {
292             len = str2.length();
293         }
294         // compares char for char, returns 1 if this-> string is larger
295
296         while (count < len)
297         {
298             if(thisString.str[count] > str2.str[count])
299             {
300                 //str1 is greater or comes later return 1
301                 result = 1;
302                 count = len; // effectivly breaks
303
304             }
305             else if (thisString.str[count] < str2.str[count])
306             {
307                 result = -1;
308                 count = len; // effectivly breaks
309             }
310             //then they must be the same char , if one is terminated here it comes first (is smaller)
311             // if the char arr is shorter than the other but equal otherwise
312             // the shorter one wins by defult
313             else if (thisString.str[count+1] == '\0')
314             {
```

```
315                 result = -1;    //compare string is larger
316                 count = len;
317
318             }
319             else if (str2.str[count+1] == '\0')
320             {
321                 result = 1;     //this string is larger, compare string comes first
322                 count = len;
323             }
324
325             count++;
326         }
327         return result;//return 0 if equal
328 }
329
330
331 const char* JCString::c_str() {
332     return this->str;
333 }
334
335 //  modifies the JCString such that it is all lower case
336 void JCString::makeLower()
337 {
338     for (int i = 0; i <= this->end; i++)
339     {
340         if (this->str[i] < 91 || this->str[i] > 64)
341         {
342             this->str[i] += 32;
343         }
344     }
345 }
346 // returns an new instance of the JCString that is lower case
347 JCString JCString::returnLower() const
348
349 {
```

```
350     JCString returnString(this->str);
351     returnString.makeLower();
352     return returnString;
353 }
354
355 // STATIC FUNCTIONS
356 char* JCString::appendCstr(const char* str1, const char* str2)
357 {
358     int count1 = 0;
359     int count2 = 0;
360     //count elements
361     for (count1; str1[count1] != '\0'; ++count1);
362     for (count2; str2[count2] != '\0'; ++count2);
363
364     char* outChar = new char[count1 + count2 + 1] ;// space for null terminal
365     //fill new char []
366     for (int i = 0; str1[i] != '\0'; ++i)
367     {
368         outChar[i] = str1[i];
369     }
370     for (int i = 0; str2[i] != '\0'; ++i)
371     {
372         outChar[count1 + i] = str2[i];
373     }
374     outChar[count1 + count2] = '\0';//terminate with null
375
376     return outChar;
377 }
378
379 int JCString::getCurrentCount()
380 {
381     return currentCount;
382 }
383 int JCString::getCreatedCount()
384 {
```

```
385     return createdCount;
386 }
387
```