```
 1
 2 // FILENAME: BigNum.cxx
 3 // This is the implementation file of the BigNum class
 4
 5
 6 #ifndef HW3_BIGNUM_CXX
 7 #define HW3_BIGNUM_CXX
 8 #include <algorithm>   // Provides copy function
 9 #include <cstdlib>
10 #include <iostream>
11 #include <string>
12 #include <cstring>
13 #include <cassert>
14 #include "BigNum.h"
15 using namespace std;
16
17 namespace HW3
18 {
19     BigNum::BigNum()
20     {
21         capacity = DEFAULT_CAPACITY;
22         digits = new unsigned int[capacity];
23         digits[0] = 0;
24         positive = true;
25         used = 1;
26     }
27
28     BigNum::BigNum(int num)
29     {
30         digits = nullptr;
31
32         if (num == 0)
33         {
34             BigNum zero;
35             *this = zero;
36         }
37
38         else
39         {
40             if (num > 0)
41             {
42                 positive = true;
43             }
44             else
45             {
46                 positive = false;
47                 num = -num;
48             }
49
50             unsigned int i = 0, temp = num;
51
52             // count the digits
53             while (temp > 0)
54             {
55                 temp = temp/10;
56                 i++;
57             }
58
```

```
59              capacity = i;
60
61              digits = new unsigned int[capacity];
62
63              temp = num;
64
65              for (used = 0; used < i; ++used)
66              {
67                  digits[used] = temp % 10;
68                  temp /= 10;
69              }
70          }
71      }
72
73      // Constructor that receives a string; leading 0's will be ignored
74      BigNum::BigNum(const string& strin)
75      {
76          digits = nullptr;
77
78          int len = strin.length();
79
80          if (len == 0)
81          {
82              BigNum zero;
83              *this = zero;
84              return;
85          }
86
87          used = len;
88          positive = true;
89
90          int i = 0;
91
92          if(strin[i] == '-')
93          {
94              positive = false;
95              i = 1;
96              used--;
97          }
98          else if(strin[i] == '+')
99          {
100             i = 1;
101             used--;
102         }
103
104         capacity = used;
105
106         digits = new unsigned int[capacity];
107
108         for(unsigned int k = 0; k < used; ++k)
109         {
110             digits[used - k - 1] = strin[i++] - '0';
111         }
112
113         if (used == 1 && digits[0] == 0)
114             positive = true;
115
116         trim();
```

```
117     }
118
119     BigNum::BigNum(const BigNum& anotherBigNum)
120     {
121         digits = nullptr;
122
123         // makes operator = do the work; use that function if you use this
124         *this = anotherBigNum;
125     }
126
127     BigNum::~BigNum()
128     {
129         delete [] digits;
130     }
131
132     // assume doubling is done before passing in n
133     void BigNum::resize(unsigned int n)
134     {
135         unsigned int *largerArray;
136
137         if (n < used) return; // Can't allocate less than we are using
138
139         capacity = n;
140         largerArray = new unsigned int[capacity];
141
142         copy(digits, digits + used, largerArray);
143
144         delete [] digits;
145         digits = largerArray;
146     }
147
148     BigNum& BigNum::operator=(const BigNum& anotherBigNum)
149     {
150         if (this == &anotherBigNum)  return *this;
151
152         if (digits != nullptr)
153             delete [] digits;
154
155         capacity = anotherBigNum.capacity;
156         digits = new unsigned int[capacity];
157
158         positive = anotherBigNum.positive;
159         used = anotherBigNum.used;
160         copy(anotherBigNum.digits, anotherBigNum.digits + used, digits);
161
162         return *this;
163     }
164
165
166     BigNum& BigNum::operator+=(const BigNum& addend)
167     {
168         return *this;
169     }
170
171     BigNum& BigNum::operator-=(const BigNum& subtractand)
172     {
173         return *this;
174     }
```

```
175
176     BigNum& BigNum::operator*=(const BigNum& multiplicand)
177     {
178         return *this;
179     }
180
181     BigNum& BigNum::operator/=(const BigNum& divisor)
182     {
183         return *this;
184     }
185
186     BigNum& BigNum::operator%=(const BigNum& divisor)
187     {
188         return *this;
189     }
190
191     BigNum& BigNum::operator++()
192     {
193         return *this;
194     }
195
196     BigNum& BigNum::operator--()
197     {
198         return *this;
199     }
200
201     BigNum& BigNum::diff(const BigNum& a, const BigNum& b)
202     {
203         return *this;
204     }
205
206
207     BigNum& BigNum::mult(const BigNum& a, const BigNum& b)
208     {
209         return *this;
210     }
211
212     BigNum& BigNum::sum(const BigNum& a, const BigNum& b)
213     {
214         return *this;
215     }
216
217     BigNum operator+(const BigNum& a, const BigNum& b)
218     {
219         BigNum result = 0;
220         return result;
221     }
222
223
224     BigNum operator-(const BigNum& a, const BigNum& b)
225     {
226         BigNum result = 0;
227         return result;
228     }
229
230
231     BigNum operator*(const BigNum& a, const BigNum& b)
232     {
```

```
233         BigNum result = 0;
234         return result;
235     }
236
237
238     BigNum operator / (const BigNum& a, const BigNum& b)
239     {
240         BigNum result = a;
241         return result;
242     }
243
244
245     BigNum operator%(const BigNum& a, const BigNum& b)
246     {
247         BigNum result;
248         return result;
249     }
250
251     bool operator>(const BigNum& a, const BigNum& b)
252     {
253         if (a.positive == true && b.positive == false) return true;
254
255         else if (a.positive == false && b.positive == true) return false;
256
257         else
258         {
259             if (a.used > b.used)
260             {
261                 if (a.positive == true) return true;
262                 else return false;
263             }
264
265             else if (a.used < b.used)
266             {
267                 if (a.positive == true) return false;
268                 else return true;
269             }
270
271             else
272             {
273                 for (unsigned int i = 0; i < a.used; ++i)
274                 {
275                     if (a.digits[a.used - 1 - i] < b.digits[b.used - 1 - i])
276                     {
277                         if(a.positive == true) return false;
278                         else return true;
279                     }
280                     if (a.digits[a.used - 1 - i] > b.digits[b.used - 1 - i])
281                     {
282                         if(a.positive == true) return true;
283                         else return false;
284                     }
285                 }
286             }
287         }
288         return false;
289     }
290
```

```
291
292     bool operator>=(const BigNum& a, const BigNum& b)
293     {
294         return ((a > b) || (a == b));
295     }
296
297
298     bool operator<(const BigNum& a, const BigNum& b)
299     {
300         return !(a >= b);
301     }
302
303
304     bool operator<=(const BigNum& a, const BigNum& b)
305     {
306         return !(a > b);
307     }
308
309
310     bool operator==(const BigNum& a, const BigNum& b)
311     {
312         if ((a.positive != b.positive) || (a.used != b.used))
313             return false;
314
315         for (unsigned int i = 0; i < a.used; i++)
316         {
317             if (a.digits[a.used - 1 - i] != b.digits[b.used - 1 - i])
318                 return false;
319         }
320
321         return true;
322     }
323
324
325     bool operator!=(const BigNum& a, const BigNum& b)
326     {
327         return !(a == b);
328     }
329
330     // trim leading zeros
331     void BigNum::trim()
332     {
333         while (used > 1 && digits[used-1] == 0)
334             used--;
335     }
336
337     std::ostream& operator<<(std::ostream &os, const BigNum& bignum)
338     {
339         unsigned int i = 0;
340         unsigned int j = 0;
341
342         if (bignum.positive == false)  os << '-';
343
344         for (i=0; i<bignum.used; ++i)
345         {
346             os << bignum.digits[bignum.used - i - 1];
347             if (j < 60) ++j;
348             else
```

```
349                 {
350                     os << endl;
351                     j = 0;
352                 }
353             }
354 
355             return os;
356         }
357 
358         std::istream& operator>>(std::istream &is, BigNum& bignum)
359         {
360             string str;
361             is >> str;
362 
363             BigNum temp = str;
364             bignum = temp;
365             return is;
366         }
367 
368         BigNum factorial(const BigNum& a)
369         {
370             BigNum result;
371             return result;
372         }
373       }
374 
375 
376 #endif
377 
378 
379 
380 
```