```cpp
/*************************************************************
*                                                            *
*         A LARGE NUMBER PROGRAM BY JAMES WASHINGTON         *
*                                                            *
*************************************************************/

#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>

using namespace std;

int MAX_DIGITS = 14;
int MAX_NUMS = 11;

void outputHeader();
int stringToInt(string);
string intToString(int);
void reverseString(string&);
string numIsValid(string);
void numOfNumValid(string&);
void ifSameSize(string[], int, bool&);
int getMax(string[], int, int&);
void formatNums(string[], int);
int* getNumStart(string[], int);
string* getNumOfNums(int&);
void getNumbers(string[], int);
string addNumbers(string[], int);
string* multiplyNumbers(string[], int, string&, int&);
void deformatNumbers(string[], int);
void addCommas(string[], int);
void addCommasAns(string&);
void isFormatted(string[], int, string&);
void outputAdd(string[], int, string, bool);
void outputMult(string[], int, string[], int, string);

int main()
{
    outputHeader();

    int size;
    string addAns;
    string multAns;
    int msize;
    string* nums = getNumOfNums(size);
    getNumbers(nums, size);

    addAns = addNumbers(nums, size);
    string* work = multiplyNumbers(nums, size, multAns, msize);

    outputAdd(nums, size, addAns, false);
```

```
53      outputMult(nums, size, work, msize, multAns);
54  }
55
56  /*************************************************************
57  *       THIS FUNCTION OUTPUTS HEADER FOR THE PROGRAM         *
58  *                                                            *
59  *************************************************************/
60
61  void outputHeader()
62  {
63      cout << "/*************************************************************\ " << endl;
64      cout << "*                                                             * " << endl;
65      cout << "*                      LARGE NUMBERS                          * " << endl;
66      cout << "*                                                             * " << endl;
67      cout << "\*************************************************************/ " << endl << endl;
68
69      cout << "WELCOME USER!" << endl << endl;
70      cout << "# TO GET STARTED ENTER HOW MANY NUMBERS YOU WANT." << endl;
71      cout << "# NEXT ENTER THE NUMBERS AND THE CALULATIONS WILL BE MADE." << endl;
72      cout << "ENJOY THE PROGRAM!!!" << endl << endl;
73  }
74
75  /*************************************************************
76  *       THIS FUNCTION CONVERTS A STRING TO AN INT             *
77  *                                                            *
78  *************************************************************/
79
80  int stringToInt(string str)
81  {
82      int num = 0;
83      int decimal;
84      int exponent = 0;
85
86      for (int i = str.length() - 1; i >= 0; i--)
87      {
88          decimal = 1;
89
90          for (int i = 0; i < exponent; i++)
91          {
92              decimal *= 10;
93          }
94
95          int digit = str[i] - '0';
96          num += digit * decimal;
97          exponent++;
98      }
99
```

```
100     return num;
101 }
102
103 /************************************************************
104 *       THIS FUNCTION CONVERTS A STRING TO AN INT            *
105 *                                                           *
106 ************************************************************/
107
108 string intToString(int num)
109 {
110     stringstream ss;
111     ss << num;
112     string str = ss.str();
113
114     return str;
115 }
116
117 /************************************************************
118 *            THIS FUNCTION REVERSES A STRING                *
119 *                                                           *
120 ************************************************************/
121
122 void reverseString(string& str)
123 {
124     char temp;
125
126     for (int i = 0; i < str.length() / 2; i++)
127     {
128         temp = str[i];
129         str[i] = str[(str.length() - 1) - i];
130         str[(str.length() - 1) - i] = temp;
131     }
132 }
133
134 /************************************************************
135 *       THIS FUNCTION VERIFIES THE NUMBERS INPUTTED          *
136 *                                                           *
137 ************************************************************/
138
139 string numIsValid(string num)
140 {
141     int digit;
142     string dig;
143     bool valid = false;
144     bool digitValid;
145
146     while (!valid)
147     {
148         if (num.length() > 1)
149         {
150             int count = 0;
151
```

```
152             while (num[count] == '0' && count < num.length() - 1)
153             {
154                 count++;
155             }
156 
157             num.erase(0, count);
158         }
159 
160         digitValid = true;
161 
162         if (num.size() > MAX_DIGITS || num == "")
163         {
164             cout << "Invalid input, try again >> ";
165             getline(cin, num);
166             cout << endl;
167             valid = false;
168             digitValid = false;
169         }
170         else
171         {
172             int count = 0;
173 
174             while (digitValid && count < num.size())
175             {
176                 dig = num[count];
177                 digit = stringToInt(dig);
178 
179                 if (digit < 0 || digit > 9)
180                 {
181                     cout << "Invalid input, try again >> ";
182                     getline(cin, num);
183                     cout << endl;
184                     digitValid = false;
185                     valid = false;
186                 }
187 
188                 count++;
189             }
190         }
191 
192         valid = digitValid;
193     }
194 
195     return num;
196 }
197 
198 /*************************************************************
199 *   THIS FUNCTION VARIFIES THE AMOUNT OF NUMBERS INPUTTED    *
200 *                                                            *
201 *************************************************************/
202 
203 void numOfNumValid(string& input)
```

```
204 {
205     bool valid = false;
206     int num;
207
208     while (!valid)
209     {
210         num = stringToInt(input);
211
212         if (num < 2 || num > MAX_NUMS)
213         {
214             cout << "Invalid input, try again >> ";
215             getline(cin, input);
216             cout << endl;
217             valid = false;
218         }
219         else
220         {
221             valid = true;
222         }
223     }
224 }
225
226 /*************************************************************
227 *           THIS FUNCTION CHECKS IF ALL THE NUMBERS          *
228 *                     ARE THE SAME SIZE                      *
229 *************************************************************/
230
231 void ifSameSize(string numbers[], int size, bool& sameSize)
232 {
233     int count = 0;
234
235     while (sameSize && count < size - 1)
236     {
237         sameSize = (numbers[count].length() == numbers[count + 1].length());
238         count++;
239     }
240 }
241
242 /*************************************************************
243 *           THIS FUNCTION GETS THE INDEX AND LENGTH          *
244 *                   OF THE BIGGEST NUMBER                    *
245 *************************************************************/
246
247 int getMax(string numbers[], int size, int& max)
248 {
249     int index = 0;
250     max = numbers[0].length();
251
252     for (int i = 0; i < size; i++)
253     {
254         if (numbers[i].length() > max)
255         {
```

```
256             max = numbers[i].length();
257             index = i;
258         }
259     }
260
261     return index;
262 }
263
264 /************************************************************
265 *        THIS FUNCTION ADDS SPACES BEFORE EACH NUMBER        *
266 *                     SO THEIR ALIGNED                       *
267 ************************************************************/
268
269 void formatNums(string numbers[], int size)
270 {
271     bool sameSize = true;
272
273     ifSameSize(numbers, size, sameSize);
274
275     if (!sameSize)
276     {
277         int max;
278         int index = getMax(numbers, size, max);
279
280         for (int i = 0; i < size; i++)
281         {
282             for (int j = 0; j < max + 1; j++)
283             {
284                 if (numbers[i].length() != numbers[index].length())
285                 {
286                     numbers[i] = " " + numbers[i];
287                 }
288             }
289         }
290     }
291 }
292
293 /************************************************************
294 *        THIS FUNCTION FINDS WHERE THE NUMBERS START         *
295 *                       FOR EACH NUMBER                      *
296 ************************************************************/
297
298 int* getNumStart(string numbers[], int size)
299 {
300     int numStart;
301     int* numStartArr = new int[size];
302
303     for (int i = 0; i < size; i++)
304     {
305         numStart = 0;
306         string strChar(1, numbers[i][0]);
307
```

```
308         while (strChar == " ")
309         {
310             numStart++;
311             strChar = numbers[i][numStart];
312         }
313
314         numStartArr[i] = numStart;
315     }
316
317     return numStartArr;
318 }
319
320 /************************************************************
321 *        THIS FUNCTION GETS THE AMOUNT OF NUMBERS           *
322 *                                                           *
323 ************************************************************/
324
325 string* getNumOfNums(int& numOfNums)
326 {
327     string input;
328
329     cout << "How many numbers >> ";
330     getline(cin, input);
331     cout << endl;
332
333     numOfNumValid(input);
334
335     numOfNums = stringToInt(input);
336     string* numbers = new string[numOfNums];
337
338     return numbers;
339 }
340
341 /************************************************************
342 *            THIS FUNCTION GETS ALL THE NUMBERS             *
343 *                                                           *
344 ************************************************************/
345
346 void getNumbers(string numbers[], int size)
347 {
348     string input;
349     int index = 0;
350
351     for (int i = 0; i < size; i++)
352     {
353         cout << "Enter number " << i + 1 << " >> ";
354         getline(cin, input);
355         cout << endl;
356
357         numbers[i] = numIsValid(input);
358     }
359
```

```
360     formatNums(numbers, size);
361 }
362
363 /*************************************************************
364 *       THIS FUNCTION ADDS THE NUMBERS DIGIT BY DIGIT        *
365 *                                                            *
366 *************************************************************/
367
368 string addNumbers(string numbers[], int size)
369 {
370     int sum = 0;
371     int carry = 0;
372     string add;
373     string result;
374     int max = numbers[0].length();
375
376     for (int i = max - 1; i >= 0; i--)
377     {
378
379         sum = 0;
380         sum += carry;
381         carry = 0;
382
383         for (int j = 0; j < size; j++)
384         {
385             string strChar(1, numbers[j][i]);
386
387             if (strChar != " ")
388             {
389                 sum += stringToInt(strChar);
390             }
391         }
392
393         add = intToString(sum);
394
395         if (max == 1)
396         {
397             result += add;
398         }
399         else if (i == 0)
400         {
401             reverseString(add);
402             result += add;
403         }
404         else
405         {
406             result += add[add.size() - 1];
407         }
408
409         carry = stringToInt(add.substr(0, add.size() - 1));
410     }
411
```

```
412     if (max != 1)
413     {
414         reverseString(result);
415     }
416
417     return result;
418 }
419
420 /*************************************************************
421 *        THIS FUNCTION MULTIPLIES THE FIRST AND LAST         *
422 *                   NUMBER DIGIT BY DIGIT                    *
423 *************************************************************/
424
425 string* multiplyNumbers(string numbers[], int size, string& ans, int& msize)
426 {
427     string* multWork;
428     int* numStart;
429     int product;
430     int carry = 0;
431     string multiplier;
432     string multiplicand;
433     string multiply;
434     string result;
435     int count = 0;
436     bool swap = false;
437
438     numStart = getNumStart(numbers, size);
439
440     if ((numbers[size - 1].length() - numStart[size - 1]) > (numbers[0].length()
        - numStart[0]))
441     {
442         string temp1 = numbers[0];
443         numbers[0] = numbers[size - 1];
444         numbers[size - 1] = temp1;
445
446         int temp2 = numStart[0];
447         numStart[0] = numStart[size - 1];
448         numStart[size - 1] = temp2;
449         swap = true;
450     }
451
452     multWork = new string[numbers[size - 1].length() - numStart[size - 1]];
453     msize = numbers[size - 1].length() - numStart[size - 1];
454
455     for (int i = numStart[size - 1]; i < numbers[size - 1].length(); i++)
456     {
457         multiplier = numbers[size - 1][i];
458         result = "";
459
460         for (int j = numbers[0].length() - 1; j >= numStart[0]; j--)
461         {
462             if (numbers[0][j] != ' ')
```

```
463             {
464                 product = 0;
465                 product += carry;
466                 carry = 0;
467                 multiplicand = numbers[0][j];
468 
469                 if (i < numbers[size - 1].length() - 1 && j == numbers[0].length
                      () - 1)
470                 {
471                     for (int k = 0; k < numbers[size - 1].length() - i - 1; k++)
472                     {
473                         result += "0";
474                     }
475                 }
476 
477                 product += stringToInt(multiplier) * stringToInt
                      (multiplicand); //cout << product << " " << multiplier << " "
                      << j << " " << multiplicand << endl;
478                 multiply = intToString(product);
479 
480                 if (multiply.length() == 1 || j == numStart[0])
481                 {
482                     result = multiply + result;
483                 }
484                 else
485                 {
486                     carry = stringToInt(multiply.substr(0, multiply.size() - 1));
487                     result = multiply[multiply.size() - 1] + result;
488                 }
489             }
490         }
491 
492         multWork[count] = result;
493         count++;
494     }
495 
496     formatNums(multWork, msize);
497     ans = addNumbers(multWork, msize);
498 
499     return multWork;
500 }
501 
502 /************************************************************
503 *     THIS FUNCTION DELETES THE SPACE BEFORE EACH NUMBER     *
504 *                                                           *
505 ************************************************************/
506 
507 void deformatNumbers(string numbers[], int size)
508 {
509     int count;
510 
511     for (int i = 0; i < size; i++)
```

```
512     {
513         count = 0;
514
515         while (numbers[i][count] == ' ')
516         {
517             numbers[i].erase(0, 1);
518         }
519     }
520 }
521
522 /*************************************************************
523 *         THIS FUNCTION ADDS COMMAS TO EACH NUMBER           *
524 *                                                            *
525 *************************************************************/
526
527 void addCommas(string numbers[], int size)
528 {
529     int pos;
530
531     for (int i = 0; i < size; i++)
532     {
533         if (numbers[i].length() > 3)
534         {
535             pos = numbers[i].length() - 3;
536
537             while (pos > 0)
538             {
539                 numbers[i].insert(pos, ",");
540                 pos -= 3;
541             }
542         }
543     }
544 }
545
546 /*************************************************************
547 *           THIS FUNCTION ADDS COMMAS TO THE ANSWER          *
548 *                                                            *
549 *************************************************************/
550
551 void addCommasAns(string& ans)
552 {
553     int pos = ans.length() - 3;
554
555     while (pos > 0)
556     {
557         ans.insert(pos, ",");
558         pos -= 3;
559     }
560 }
561
562 /*************************************************************
563 *           THIS FUNCTION CHECKS IF NUMBERS NEED             *
```

```
564 *                COMMAS AND/OR SPACES DELETED                 *
565 ************************************************************/
566
567 void isFormatted(string numbers[], int size, string& ans)
568 {
569     bool hasCommas = false;
570     bool moreThanThree = false;
571     bool isDeformatted = true;
572     bool ansHasCommas = false;
573     bool ansMoreThanThree = (ans.length() > 3);
574
575     for (int i = 0; i < size; i++)
576     {
577         if (numbers[i][0] == ' ')
578         {
579             isDeformatted = false;
580         }
581     }
582
583     for (int i = 0; i < size; i++)
584     {
585         if (numbers[i].length() > 3)
586         {
587             moreThanThree = true;
588             for (int j = 0; j < numbers[i].length(); j++)
589             {
590                 if (numbers[i][j] == ',')
591                 {
592                     hasCommas = true;
593                 }
594             }
595         }
596     }
597
598     for (int i = 0; i < ans.length(); i++)
599     {
600         if (ans[i] == ',')
601         {
602             ansHasCommas = true;
603         }
604     }
605
606     if (!isDeformatted)
607     {
608         deformatNumbers(numbers, size);
609     }
610
611     if (moreThanThree && !hasCommas)
612     {
613         addCommas(numbers, size);
614     }
615
```

```
616     if (ansMoreThanThree && !ansHasCommas)
617     {
618         addCommasAns(ans);
619     }
620 }
621
622 /************************************************************
623 *            THIS FUNCTION OUTPUTS THE ADDITION             *
624 *                                                           *
625 ************************************************************/
626
627 void outputAdd(string numbers[], int size, string ans, bool isMultiply)
628 {
629     if (!isMultiply)
630     {
631         cout << "The sum of: " << endl << endl;
632     }
633
634     isFormatted(numbers, size, ans);
635
636     int maxLen = ans.length() + 3;
637
638     for (int i = 0; i < size; i++)
639     {
640         if (i == size - 1)
641         {
642             cout << "+) " << right << setw(maxLen - 3) << numbers[i] << endl;
643         }
644         else
645         {
646             cout << right << setw(maxLen) << numbers[i] << endl;
647         }
648     }
649
650     for (int i = 0; i < maxLen; i++)
651     {
652         cout << "-";
653     }
654
655     cout << endl << right << setw(maxLen) << ans << endl << endl << endl;
656 }
657
658 /************************************************************
659 *         THIS FUNCTION OUTPUTS THE MULTIPLICATION          *
660 *                                                           *
661 ************************************************************/
662
663 void outputMult(string numbers[], int size, string multWork[], int msize, string
      ans)
664 {
665     isFormatted(numbers, size, ans);
666
```

```
667     int multiplicandLen = numbers[0].length();
668     int multiplierLen = numbers[size - 1].length();
669     int maxLen = ans.length() + 3;
670
671     cout << "The product of:" << endl << endl;
672
673     cout << right << setw(maxLen) << numbers[0] << endl;
674     cout << "*) " << right << setw(maxLen - 3) << numbers[size - 1] << endl;
675
676     for (int i = 0; i < maxLen; i++)
677     {
678         cout << "-";
679     }
680
681     cout << endl;
682
683     outputAdd(multWork, msize, ans, true);
684 }
```