# CSC 216 Portfolio 1

Nicolas Nytko

Sep 9, 2016

# Contents

# 1 Homework

## 1.1 Linked Lists

### 1.1.1 Problem R-3.7

*Give an algorithm for finding the penultimate (second to last) node in a singly linked list where the last element is indicated by a null next link.*

Listing 1: ../hw/r-3.7.cpp

```cpp
node* getPenultimateNode( node* pStart )
{
    // Check to see if our starting node is NULL or not

    if ( pStart != NULL )
    {
        // Make sure we have more than 1 node

        if ( pStart->next == NULL )
            return pStart;

        node* pCurrent = pStart;

        while ( pCurrent->next->next != NULL ||
                ( pCurrent->next->next == NULL && pCurrent->next != NULL ) )
        {
            pCurrent = pCurrent->next;
        }

        return pCurrent;
    }
    else
    {
        return NULL;
    }
}
```

### 1.1.2 Problem R-3.10

*Describe a nonrecursive function for finding, by link hopping, the middle node of a doubly linked list with header and trailer sentinels. (Note: This function must only use link hopping; it cannot use a counter.) What is the running time of this function?*

Listing 2: ../hw/r-3.10.cpp

```cpp
node* getMiddleNode( list* pList )
{
    if ( pList == NULL )
        return NULL;

    // Have two nodes, one that starts at the beginning of the list and one that
    //     starts at the end

    node* pStart, pEnd;
    pStart = list->start;
    pEnd = list->end;

```

```
12        // Loop until the nodes "intersect" and point to the same data
13
14        while ( !( pStart == pEnd || ( pStart->next == pEnd && pEnd->prev == pStart ) )
              )
15        {
16             pStart = pStart->next;
17             pEnd = pEnd->prev;
18        }
19
20        return pStart;
21  }
```

### 1.1.3   Problem C-3.3

> *Let A be an array of size $n \geq 2$ containing integers from $1$ to $n - 1$, inclusive, with exactly one repeated. Describe a fast algorithm for finding the integer in A that is repeated.*

Listing 3: ../hw/c-3.3.cpp

```
1   int getRepeatNumber( int* pArray, size_t nLength )
2   {
3        bool bFound = false;
4        int nRepeat = -1;
5
6        for ( size_t i=0; i < nLength && !bFound; i++ )
7        {
8             for ( size_t j=i+1; j < nLength && !bFound; j++ )
9             {
10                 if ( pArray[i] == pArray[j] )
11                 {
12                      nRepeat = pArray[i];
13                      bFound = true;
14                 }
15            }
16        }
17
18        return nRepeat;
19  }
```

### 1.1.4   Problem C-3.4

> *Let B be an array of size $n \geq 6$ containing integers from $1$ to $n - 5$, inclusive, with exactly five repeated. Describe a good algorithm for finding the five integers in B that are repeated.*

Sort B. Create a variable that holds how many times the current value has been repeated, and a variable to hold what the last value was. Loop through the array and increment the repeat variable every time the value has been repeated, and reset it to 0 when a new value is introduced. End when the repeat value reaches 5.

### 1.1.5   Problem C-3.5

> *Suppose you are designing a multi-player game that has $n \geq 1000$ players, numbered $1$ to $n$, interacting in an enchanted forest. The winner of this game is the first player who can meet all the other players at least once (ties are allowed Assuming that there is a function*

> *meet(i, j), which is called each time a player i meets a player j (with i ≠ j ), describe a way*
> *to keep track of the pairs of meeting players and who is the winner.*

Create a two dimensional boolean array with the x axis corresponding to the index of the player, and the y axis corresponding to whether that person was met. So to see if player x has met player y, check array index `p[x][y]`. If a full column is filled, then that player has met everyone.

### 1.1.6   Problem C-3.8

> *Describe a good algorithm for concatenating two singly linked lists L and M, with header*
> *sentinels, into a single list L' that contains all the nodes of L followed by all the nodes of*
> *M. Create a new list L' and copy all of the nodes from L into it. At the end of L', copy all*
> *of the nodes of M.*

Copy all of the elements from L into L'. Start copying M at the last node of L' after the header sentinel and continue until the null pointer.

### 1.1.7   Problem C-3.9

> *Give a fast algorithm for concatenating two doubly linked lists L and M, with header and*
> *trailer sentinel nodes, into a single list L.*

Remove L's trailing sentinel and M's header sentinel and save L's last node and M's first node, we will call them L' and M' respectively. Set L' next to M', the previous of M' to L. The lists are now concatenated. Copy all the nodes from L into L'. Copy the nodes from M into the end of L'. Set the next

### 1.1.8   Problem C-3.10

> *Describe in detail how to swap two nodes x and y (and not just their contents) in a singly*
> *linked list L given references only to x and y. Repeat this exercise for the case when L is a*
> *doubly linked list. Which algorithm takes more time?*

For a singly linked list: loop through until the nodes before x and y are found, label them x' and y'. Set x' next node to y, and y next to x original next. Set y' next node to x, and x next node to y original next.

For a doubly linked list: do the same thing except time is not needed to loop through and find the previous nodes. Don't forget to set the previous node values.

### 1.1.9   Problem C-3.11

> *Describe in detail an algorithm for reversing a singly linked list L using only a constant*
> *amount of additional space and not using any recursion.*

Define n to be the size of the linked list. Loop `n-1` times with iterator i starting at 0, and bring node `n-i` backwards by swapping it `n-i` times with the node previous. Save the previous node as a pointer variable.

### 1.1.10   Problem C-3.22

> *Suppose you are given two circularly linked lists, L and M, that is, two lists of nodes such*
> *that each node has a nonnull next node. Describe a fast algorithm for telling if L and M are*
> *really the same list of nodes but with different (cursor) starting points.*

Create two node pointers, lowercase l and m, and set them to point to the head of L and M respectively. Loop through the list until l is equal to M and m is equal to L (they are the same list), or l is equal to L and m is equal to M (the whole list has been parsed and L and M are not the same list).

### 1.1.11   Problem C-3.23

> *Given a circularly linked list L containing an even number of nodes, describe how to split L into two circularly linked lists of half the size.*

Given a circularly linked list L, define M to be the length of L divided by two. Create two node pointers, x and y, and have them point to the beginning of L and the node at M, respectively. For the last node in the first half (M-1), set its next node to the first node. For the first node in the first half (0), set its previous node to M-1. For node M, set its previous node to the last node in the list. Set the last node's next to M.

# 2  Projects

## 2.1  Encryption Cipher

Name:          Nicolas Nytko                                    Course: CSC216

Activity:      Encryption Cipher

Level:         5

Description:   Write a program that can perform encryption and decryption using an arbitrary substitution cipher. In this case, the encryption array is a random shuffling of the letters in the alphabet. Your program should generate a random encryption array, its corresponding decryption array, and use these to encode and decode a message. Allow for the saving and loading of encrypted messages by storing the 26 letter encryption key amongst the n encoded characters of the message. Note that for an $n$ character message, there will be $n + 1$ slots amongst them. For a 1-character message, for instance, there is a slot before the character and a slot after the character. For a 2-character message, there are slots before and after the first character and after the second character. And so on... Make sure the extra 26 mod $(n + 1)$ letters from the encryption key are located carefully to make the spread nice and even. (Note that when the message is longer than 25 characters, each letter of the key is alone by itself and, in fact, you are spreading the characters of the message amongst the 27 slots around the key values now.)

### 2.1.1  Compiler Environment

Listing 4: environment

```
 1        tex git:(master)        pwd
 2  /Users/nicolas/Git/portfolio1/tex
 3        tex git:(master)        uname −a
 4  Darwin Nicolass−MacBook−Pro.local 16.0.0 Darwin Kernel Version 16.0.0: Mon Aug 29
        17:56:20 PDT 2016; root:xnu−3789.1.32~3/RELEASE_X86_64 x86_64
 5        tex git:(master)        clang −−version
 6  Apple LLVM version 8.0.0 (clang−800.0.38)
 7  Target: x86_64−apple−darwin16.0.0
 8  Thread model: posix
 9  InstalledDir: /Library/Developer/CommandLineTools/usr/bin
10        ~ harper_cpp −−version
11  This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:
12
13  Configured with: −−prefix=/Library/Developer/CommandLineTools/usr −−with−gxx−include
        −dir=/usr/include/c++/4.2.1
14  Apple LLVM version 8.0.0 (clang−800.0.38)
15  Target: x86_64−apple−darwin16.1.0
16  Thread model: posix
17  InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

### 2.1.2  Source

Listing 5: ../project/cipher/Makefile

```
1  CC=g++
2  OUTPUT=cipher.out
3  INPUT=main.cpp
4  CCFLAGS=
5
6  all:
```

```
7          $(CC)  $(CCFLAGS) −std=c++14  $(INPUT) −o  $(OUTPUT)
```

Listing 6: ../project/cipher/main.cpp

```cpp
 1  #include <iostream>
 2  #include <cmath>
 3  #include <fstream>
 4  #include <cstring>
 5  #include <ctime>
 6
 7  class CipherKey
 8  {
 9  private:
10      static const unsigned long KEY_LENGTH = 26;
11      char pKey[KEY_LENGTH + 1]; // Plus one for the null terminator
12
13      unsigned long getKeyIndex( char c ) const
14      {
15          unsigned long nReturn = 0;
16
17          if ( !( c >= 'A' && c <= 'Z' ) )
18              return 0;
19
20          for ( unsigned long i=0; i < KEY_LENGTH && nReturn == 0; i++ )
21          {
22              if ( c == pKey[i] )
23                  nReturn = i;
24          }
25
26          return nReturn;
27      }
28
29  public:
30      CipherKey( )
31      {
32          std::memset( pKey, 0, KEY_LENGTH + 1 );
33      }
34
35      CipherKey( const CipherKey& pOther )
36      {
37          std::memcpy( pKey, pOther.pKey, KEY_LENGTH );
38      }
39
40      CipherKey& operator=( const CipherKey& pOther )
41      {
42          std::memcpy( pKey, pOther.pKey, KEY_LENGTH );
43
44          return *this;
45      }
46
47      void generate( unsigned int nSeed )
48      {
49          std::srand( nSeed );
50
51          for ( unsigned long i=0; i < KEY_LENGTH; i++ )
52          {
53              pKey[i] = 'A' + static_cast<char>( i );
54          }
55
```

```cpp
56              for ( unsigned long i=0; i < KEY_LENGTH; i++ )
57              {
58                  std::swap( pKey[i], pKey[ static_cast<unsigned int>( std::rand( ) ) %
                        KEY_LENGTH ] );
59              }
60          }
61
62          void setKey( const char* pNewKey )
63          {
64              if ( strlen( pNewKey ) != KEY_LENGTH )
65              {
66                  std::cerr << "CipherKey::setKey(" << pNewKey << "): new key is not "
67                            << KEY_LENGTH << "characters long." << std::endl;
68                  return;
69              }
70
71              std::strncpy( pKey, pNewKey, KEY_LENGTH + 1 );
72          }
73
74          std::string encrypt( const std::string& sEncrypt ) const
75          {
76              std::string sReturn;
77
78              for ( size_t i=0; i < sEncrypt.length( ); i++ )
79              {
80                  char c = sEncrypt[i];
81
82                  if ( std::islower( c ) )
83                      c = static_cast<char>( std::toupper( c ) );
84
85                  if ( std::isupper( c ) )
86                      sReturn += pKey[ c - 'A' ];
87              }
88
89              return sReturn;
90          }
91
92          std::string decrypt( const std::string& sDecrypt ) const
93          {
94              std::string sReturn;
95
96              for ( size_t i=0; i < sDecrypt.length( ); i++ )
97              {
98                  unsigned long nIndex = getKeyIndex( sDecrypt[i] );
99
100                 sReturn += 'A' + static_cast<char>( nIndex );
101             }
102
103             return sReturn;
104         }
105
106         unsigned int getKeyLength( ) const
107         {
108             return KEY_LENGTH;
109         }
110
111         const char* getKey( ) const
112         {
113             return pKey;
```

```cpp
114          }
115      };
116
117      class FileCipher
118      {
119      private:
120          CipherKey pCipherKey;
121          std::string sMessage;
122
123      public:
124          FileCipher( ): pCipherKey( ), sMessage( "" )
125          {
126              pCipherKey.generate( static_cast<unsigned int>( std::time( NULL ) ) );
127          }
128
129          FileCipher( const std::string sSetMessage ): pCipherKey( ), sMessage(
                  sSetMessage )
130          {
131              pCipherKey.generate( static_cast<unsigned int>( std::time( NULL ) ) );
132          }
133
134          FileCipher( const FileCipher& pOther ): pCipherKey( pOther.pCipherKey ),
135                                                  sMessage( pOther.sMessage ) { }
136
137          FileCipher& operator=( const FileCipher& pOther )
138          {
139              pCipherKey = pOther.pCipherKey;
140              sMessage = pOther.sMessage;
141
142              return *this;
143          }
144
145          std::string getMessage( ) const
146          {
147              return sMessage;
148          }
149
150          void setMessage( std::string sNewMessage )
151          {
152              sMessage = sNewMessage;
153          }
154
155          std::string encrypt( ) const
156          {
157              return pCipherKey.encrypt( sMessage );
158          }
159
160          std::string decrypt( std::string sToDecrypt )
161          {
162              return pCipherKey.decrypt( sToDecrypt );
163          }
164
165          std::string encryptWithKey( ) const
166          {
167              std::string sEncrypted = encrypt( );
168              std::string sReturn = "";
169              const char* szKey = pCipherKey.getKey( );
170
171              double nCharsPerSlot = (double) pCipherKey.getKeyLength( ) /
```

```cpp
172                     ( sMessage.length( ) + 1 );
173             unsigned long nCurrentKey = 0;
174
175             /* Do first key pass */
176
177             for ( unsigned long i=nCurrentKey; i < nCharsPerSlot; i++, nCurrentKey++ )
178             {
179                 sReturn += szKey[ nCurrentKey ];
180             }
181
182             /* Write all characters */
183
184             for ( unsigned long nChar=0; nChar < sMessage.length( ); nChar++ )
185             {
186                 sReturn += sEncrypted[nChar];
187
188                 /* Write keys after all message characters */
189
190                 for ( ; nCurrentKey < static_cast<unsigned long>( nCharsPerSlot * (
                        nChar + 2 ) ); nCurrentKey++ )
191                 {
192                     sReturn += szKey[ nCurrentKey ];
193                 }
194             }
195
196         /* Write any leftover keys */
197
198         for ( ; nCurrentKey < pCipherKey.getKeyLength( ); nCurrentKey++ )
199         {
200             sReturn += szKey[ nCurrentKey ];
201         }
202
203         return sReturn;
204     }
205
206     std::string decryptWithKey( std::string sEncrypted )
207     {
208         std::string sMsg, sKey;
209
210         unsigned long nChars = sEncrypted.length( ) - static_cast<unsigned long>(
                pCipherKey.getKeyLength( ) );
211         double nCharsPerSlot = (double) pCipherKey.getKeyLength( ) /
212             ( nChars + 1 );
213
214         /* nCurrent is position in overall string, nCurrentKey is
215          * current part of the key */
216
217         unsigned long nCurrent=0, nCurrentKey = 0;
218
219         /* Do first key pass */
220
221         for ( ; nCurrentKey < nCharsPerSlot; nCurrentKey++ )
222         {
223             sKey += sEncrypted[nCurrent++];
224         }
225
226         /* Look at all characters */
227
228         for ( size_t nMsg = 0; nMsg < nChars; nMsg++ )
```

```cpp
229                 {
230                     sMsg += sEncrypted[nCurrent++];
231
232                     for ( ; nCurrentKey < static_cast<unsigned long>( nCharsPerSlot * ( nMsg
                            + 2 ) ); nCurrentKey++ )
233                     {
234                         sKey += sEncrypted[nCurrent++];
235                     }
236                 }
237
238                 /* Check any leftover keys */
239
240                 for ( ; nCurrent < sEncrypted.length( ); nCurrent++ )
241                 {
242                     sKey += sEncrypted[nCurrent];
243                 }
244
245                 pCipherKey.setKey( sKey.c_str( ) );
246                 sMessage = decrypt( sMsg );
247
248                 return sMessage;
249         }
250
251         friend std::ostream& operator<<( std::ostream& pOutput, const FileCipher&
                pCipher )
252         {
253             pOutput << pCipher.encryptWithKey( ) << std::endl;
254
255             return pOutput;
256         }
257
258         friend std::istream& operator>>( std::istream& pInput, FileCipher& pCipher )
259         {
260             std::string sLine;
261             std::getline( pInput, sLine );
262
263             pCipher.decryptWithKey( sLine );
264
265             return pInput;
266         }
267 };
268
269 void printUsage( char* arg0 )
270 {
271     std::cout << "USAGE: " << arg0 << " -f <filename> [options]" << std::endl << std
            ::endl;
272     std::cout << "OPTIONS: " << std::endl;
273     std::cout << "    -f, --file\t\t\tFile to do operations on." << std::endl;
274     std::cout << "    -e, --encrypt <msg>\t\tEncrypts <msg> and saves it in file."
            << std::endl;
275     std::cout << "    -d, --decrypt\t\tDecrypts the message loaded from file." <<
            std::endl << std::endl;
276 }
277
278 enum ProgramStatus
279 {
280     STATUS_UNKNOWN,
281     STATUS_ENCRYPT,
282     STATUS_DECRYPT
```

```cpp
283  };
284
285  int main( int argc, char** argv )
286  {
287      std::string sFile;
288      std::string sInputMsg;
289      ProgramStatus nStatus = STATUS_UNKNOWN;
290
291      if ( argc < 2 )
292      {
293          printUsage( argv[0] );
294          return 1;
295      }
296      else
297      {
298          bool bArgsGood = true;
299          int i = 1;
300
301          while ( i < argc && bArgsGood )
302          {
303              char* szArg = argv[i];
304
305              if ( szArg[0] != '-' )
306              {
307                  bArgsGood = false;
308              }
309              else
310              {
311                  if ( szArg[1] == 'f' || std::strcmp( szArg, "--file" ) == 0 )
312                  {
313                      sFile = argv[i+1];
314                      i += 2;
315                  }
316                  if ( szArg[1] == 'e' || std::strcmp( szArg, "--encrypt" ) == 0 )
317                  {
318                      nStatus = STATUS_ENCRYPT;
319                      sInputMsg = argv[i+1];
320
321                      i += 2;
322                  }
323                  if ( szArg[1] == 'd' || std::strcmp( szArg, "--decrypt" ) == 0 )
324                  {
325                      nStatus = STATUS_DECRYPT;
326                      i++;
327                  }
328              }
329          }
330
331          if ( !bArgsGood )
332          {
333              printUsage( argv[0] );
334              return 1;
335          }
336      }
337
338      FileCipher pCipher;
339
340      if ( nStatus == STATUS_ENCRYPT )
341      {
```

13

```
342          std::ofstream fOutput;
343
344          pCipher.setMessage( sInputMsg );
345          fOutput.open( sFile, std::ofstream::out );
346
347          fOutput << pCipher;
348
349          fOutput.close( );
350      }
351      else if ( nStatus == STATUS_DECRYPT )
352      {
353          std::ifstream fInput;
354
355          fInput.open( sFile, std::ifstream::in );
356          fInput >> pCipher;
357
358          std::cout << pCipher.getMessage( ) << std::endl;
359      }
360
361      return 0;
362 }
```

### 2.1.3   Compiler Output

Listing 7: ../project/cipher/compilerout

```
1         cipher git:(master)      make CC=harper_cpp
2 harper_cpp   -std=c++14 main.cpp -o cipher.out
3 main.cpp***
```

### 2.1.4   Program Output

Listing 8: ../project/cipher/progout

```
1         cipher git:(master)     ./cipher.out
2 USAGE:  ./cipher.out -f <filename> [options]
3
4 OPTIONS:
5     -f, --file              File to do operations on.
6     -e, --encrypt <msg>          Encrypts <msg> and saves it in file.
7     -d, --decrypt          Decrypts the message loaded from file.
8
9      cipher git:(master)      ./cipher.out -f file -e helloworld
10      cipher git:(master)      ./cipher.out -f file -d
11 HELLOWORLD
12      cipher git:(master)      ./cipher.out -f file -e goodmorningjasonjames
13      cipher git:(master)      ./cipher.out -f file -d
14 GOODMORNINGJASONJAMES
```

14

# 3 Labs

## 3.1 Game Highscores

Name:         Nicolas Nytko                                                      Course: CSC216

Activity:     Game Highscores
Level:        4
Description:  P-3.4. Store 10 game highscores in a doubly-linked list.

### 3.1.1 Compiler Environment

Listing 9: environment

```
 1       tex git:(master)       pwd
 2 /Users/nicolas/Git/portfolio1/tex
 3       tex git:(master)       uname -a
 4 Darwin Nicolass-MacBook-Pro.local 16.0.0 Darwin Kernel Version 16.0.0: Mon Aug 29
      17:56:20 PDT 2016; root:xnu-3789.1.32~3/RELEASE_X86_64 x86_64
 5       tex git:(master)       clang --version
 6 Apple LLVM version 8.0.0 (clang-800.0.38)
 7 Target: x86_64-apple-darwin16.0.0
 8 Thread model: posix
 9 InstalledDir: /Library/Developer/CommandLineTools/usr/bin
10       ~ harper_cpp --version
11 This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:
12
13 Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include
      -dir=/usr/include/c++/4.2.1
14 Apple LLVM version 8.0.0 (clang-800.0.38)
15 Target: x86_64-apple-darwin16.1.0
16 Thread model: posix
17 InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

### 3.1.2 Source

Listing 10: ../lab/highscores/main.cpp

```cpp
 1 #include <iostream>
 2
 3 /* P-3.4
 4  * Write a class that maintains the top 10 scores for a game application,
 5  * implementing the add and remove functions of Section 3.1.1, but use a
 6  * doubly linked list. Your implementation of remove(i) should make the
 7  * fewest number of pointer hops to get to the game entry at index i.
 8  */
 9
10 /**
11  * High score entry containing name and score.
12  */
13
14 class GameEntry
15 {
16 private:
17     std::string name;
18     int score;
19
```

```cpp
public:
    GameEntry( const std::string& setName="", int setScore=0 ): name( setName ),
            score( setScore ) { }
    GameEntry( const GameEntry& pOther ): name( pOther.name ), score( pOther.score )
            { }

    GameEntry& operator=( const GameEntry& pOther )
    {
        name = pOther.name;
        score = pOther.score;

        return *this;
    }

    std::string getName( ) const
    {
        return name;
    }

    int getScore( ) const
    {
        return score;
    }

    void setName( const std::string& sNewName )
    {
        name = sNewName;
    }

    void setScore( int nNewScore )
    {
        score = nNewScore;
    }
};

/**
 * Linked-list node.
 */

class ListNode
{
private:
    GameEntry pData;
    class ListNode* pPrev, *pNext;

public:
    ListNode( ): pPrev( NULL ), pNext( NULL ) { }

    ListNode( const GameEntry& pSetData ): pData( pSetData ),
                                           pPrev( NULL ),
                                           pNext( NULL ) { }

    ListNode( const ListNode& pOther ): pData( pOther.pData ),
                                        pPrev( pOther.pPrev ),
                                        pNext( pOther.pNext ) { }

    ListNode& operator=( const ListNode& pOther )
    {
        pData = pOther.pData;
```

```cpp
77              pPrev = pOther.pPrev;
78              pNext = pOther.pNext;
79
80              return *this;
81          }
82
83          GameEntry& getData( )
84          {
85              return pData;
86          }
87
88          GameEntry getData( ) const
89          {
90              return pData;
91          }
92
93          friend class List;
94  };
95
96  /**
97   * Linked-list implementation class.
98   */
99
100 class List
101 {
102 private:
103     ListNode* pFirst, *pLast;
104     size_t nLength;
105
106     ListNode* getNode( size_t nIndex )
107     {
108         if ( nLength == 0 )
109             return NULL;
110
111         ListNode* pReturn;
112
113         if ( nIndex < nLength/2 )
114         {
115             pReturn = pFirst;
116
117             for ( size_t i=0; i < nIndex; i++ )
118             {
119                 pReturn = pReturn->pNext;
120             }
121         }
122         else
123         {
124             pReturn = pLast;
125
126             for ( size_t i=nLength-1; i > nIndex; i-- )
127             {
128                 pReturn = pReturn->pPrev;
129             }
130         }
131
132         return pReturn;
133     }
134
135 public:
```

```cpp
136         List( ): pFirst( NULL ), pLast( NULL ), nLength( 0 )
137         {
138             pFirst = new ListNode;
139             pLast = pFirst;
140         }
141
142         List( const List& pOther ): pFirst( pOther.pFirst ),
143                                     pLast( pOther.pLast ),
144                                     nLength( pOther.nLength ) { }
145
146         List& operator=( const List& pOther )
147         {
148             pFirst = pOther.pFirst;
149             pLast = pOther.pLast;
150             nLength = pOther.nLength;
151
152             return *this;
153         }
154
155         ~List( )
156         {
157             if ( nLength != 0 )
158             {
159                 for ( ListNode* pCurrent = pFirst;
160                       pCurrent != NULL;
161                       pCurrent = pCurrent->pNext )
162                 {
163                     delete pCurrent;
164                 }
165             }
166         }
167
168         size_t getLength( ) { return nLength; }
169
170         GameEntry& get( size_t nIndex )
171         {
172             return getNode( nIndex )->getData( );
173         }
174
175         GameEntry& operator []( const size_t nIndex )
176         {
177             return getNode( nIndex )->getData( );
178         }
179
180         void push_back( const GameEntry& pData )
181         {
182             if ( nLength == 0 )
183             {
184                 pFirst = new ListNode( pData );
185                 pLast = pFirst;
186             }
187             else
188             {
189                 ListNode* pTempNode = new ListNode( pData );
190                 pLast->pNext = pTempNode;
191                 pTempNode->pPrev = pLast;
192                 pLast = pTempNode;
193             }
194
```

```cpp
195            nLength++;
196        }
197
198        bool insert( size_t nIndex, const GameEntry& pData )
199        {
200            if ( nLength == 0 )
201            {
202                if ( nIndex != 0 )
203                    return false;
204
205                /* If theres no other nodes, call our push_back function */
206
207                push_back( pData );
208
209                return true;
210            }
211            else
212            {
213                if ( nIndex > nLength )
214                    return false; /* Fail if last node slot + 1 */
215
216                if ( nIndex == nLength )
217                {
218                    /* If we're trying to place at the last slot then call push_back */
219
220                    push_back( pData );
221                    return true;
222                }
223                else if ( nIndex == 0 )
224                {
225                    /* If we're trying to place at the beginning */
226
227                    ListNode* pSecond = pFirst;
228                    ListNode* pNewTemp = new ListNode( pData );
229
230                    pNewTemp->pNext = pSecond;
231                    pFirst = pNewTemp;
232
233                    nLength++;
234
235                    return true;
236                }
237                else
238                {
239                    /* Placing at an arbitrary point in the list */
240
241                    ListNode* pAt = getNode( nIndex );
242                    ListNode* pPrev = pAt->pPrev;
243                    ListNode* pNewTemp = new ListNode( pData );
244
245                    pNewTemp->pPrev = pPrev;
246                    pNewTemp->pNext = pAt;
247                    pAt->pPrev = pNewTemp;
248                    pPrev->pNext = pNewTemp;
249
250                    nLength++;
251
252                    return true;
253                }
```

```cpp
254                }

256            return false;
257        }

259        bool remove( size_t nIndex )
260        {
261            if ( nLength == 0 )
262            {
263                /* Can't remove when there's already nothing */

265                return true;
266            }
267            else if ( nLength == 1 )
268            {
269                delete pFirst;
270                pFirst = NULL;
271                pLast = NULL;

273                return true;
274            }
275            else
276            {
277                if ( nIndex >= nLength )
278                    return false;

280                /* If last node, remove and update the new last node */

282                if ( nIndex == nLength - 1 )
283                {
284                    ListNode* pNewEnd = pLast->pPrev;
285                    delete pLast;

287                    pLast = pNewEnd;
288                    pNewEnd->pNext = NULL;

290                    return true;
291                }
292                else
293                {
294                    /* Else, remove node and update next and previous nodes to point
295                       to each other */

297                    ListNode* pPrev, *pNext, *pCurrent;
298                    pCurrent = getNode( nIndex );
299                    pPrev = pCurrent->pPrev;
300                    pNext = pCurrent->pNext;

302                    delete pCurrent;
303                    pPrev->pNext = pNext;
304                    pNext->pPrev = pPrev;

306                    return true;
307                }
308            }
309        }
310    };

312    int main( )
```

```
313  {
314      List scores;
315
316      scores.push_back( GameEntry( "BOB", 50000 ) );
317      scores.push_back( GameEntry( "NIK", 42000 ) );
318      scores.push_back( GameEntry( "ASK", 36900 ) );
319      scores.push_back( GameEntry( "TUT", 31000 ) );
320      scores.push_back( GameEntry( "DAN", 20000 ) );
321
322      scores.insert( 0, GameEntry( "LOL", 65000 ) );
323      scores.insert( 6, GameEntry( "SUX", 100 ) );
324      scores.insert( 6, GameEntry( "BWA", 500 ) );
325      scores.insert( 6, GameEntry( "ASD", 1000 ) );
326      scores.insert( 6, GameEntry( "DSF", 2000 ) );
327
328      for ( size_t i=0; i < scores.getLength( ); i++ )
329      {
330          std::cout << i << ": " << scores[i].getName( ) << ": " << scores[i].getScore
                 ( ) << std::endl;
331      }
332
333      return 0;
334  }
```

### 3.1.3 Compiler Output

Listing 11: ../lab/highscores/compilerout

```
1        highscores git:(master)      make CC=harper_cpp
2  harper_cpp  -std=c++14 main.cpp -o highscores.out
3  main.cpp***
```

### 3.1.4 Program Output

Listing 12: ../lab/highscores/progout

```
1        highscores git:(master)      ./highscores.out
2  0: LOL: 65000
3  1: BOB: 50000
4  2: NIK: 42000
5  3: ASK: 36900
6  4: TUT: 31000
7  5: DAN: 20000
8  6: DSF: 2000
9  7: ASD: 1000
10 8: BWA: 500
11 9: SUX: 100
```

## 3.2 Matrix Class

Name:        Nicolas Nytko                                    Course: CSC216

Activity:    Matrix Class
Level:       3
Description: P-3.2. Matrix class with multiplication and addition operators.

### 3.2.1 Compiler Environment

Listing 13: environment

```
 1        tex git:(master)        pwd
 2  /Users/nicolas/Git/portfolio1/tex
 3        tex git:(master)        uname -a
 4  Darwin Nicolass-MacBook-Pro.local 16.0.0 Darwin Kernel Version 16.0.0: Mon Aug 29
        17:56:20 PDT 2016; root:xnu-3789.1.32~3/RELEASE_X86_64 x86_64
 5        tex git:(master)        clang --version
 6  Apple LLVM version 8.0.0 (clang-800.0.38)
 7  Target: x86_64-apple-darwin16.0.0
 8  Thread model: posix
 9  InstalledDir: /Library/Developer/CommandLineTools/usr/bin
10        ~ harper_cpp --version
11  This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:
12
13  Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include
        -dir=/usr/include/c++/4.2.1
14  Apple LLVM version 8.0.0 (clang-800.0.38)
15  Target: x86_64-apple-darwin16.1.0
16  Thread model: posix
17  InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

### 3.2.2 Source

Listing 14: ../lab/matrix/main.cpp

```cpp
 1  #include <iostream>
 2  #include "matrix.hpp"
 3
 4  int main( )
 5  {
 6      Matrix<3,2> a = { 1, 2, 3, 4, 5, 6 };
 7      Matrix<2,3> b = { 7, 8, 9, 10, 11, 12, 13 };
 8      Matrix<2,3> c = { 14, 15, 16, 17, 18, 19 };
 9
10      std::cout << "Matrices: " << std::endl;
11      std::cout << "a:    " << a << std::endl << "b:    " << b << std::endl << "c:    "
            << c << std::endl << std::endl;
12
13      auto product = a * b;
14      auto sum = b + c;
15
16      std::cout << "a*b: " << product << std::endl;
17      std::cout << "b+c: " << sum << std::endl;
18
19      return 0;
20  }
```

Listing 15: ../lab/matrix/matrix.hpp

```cpp
 1  #ifndef LIB_MATRIX_HPP
 2  #define LIB_MATRIX_HPP
 3
 4  #include <fstream>
 5
 6  /**
```

```cpp
 7    * Template class for an nRows by nCols matrix.
 8    */
 9
10   template <short nRows, short nCols>
11   class Matrix
12   {
13   private:
14       double dData[nRows*nCols];
15
16   public:
17       Matrix( )
18       {
19           for ( size_t i=0; i < nRows*nCols; i++ )
20           {
21               dData[i] = 0.0;
22           }
23       }
24
25       Matrix( std::initializer_list<double> dList )
26       {
27           for ( size_t i=0; i < dList.size( ) && i < nRows*nCols; i++ )
28           {
29               dData[i] = *( dList.begin( ) + i );
30           }
31       }
32
33       constexpr unsigned long getArea( ) const { return nRows*nCols; }
34       constexpr short getRows( ) const { return nRows; }
35       constexpr short getCols( ) const { return nCols; }
36       constexpr short getHeight( ) const { return nRows; }
37       constexpr short getWidth( ) const { return nCols; }
38
39       double& operator [] ( const size_t nIndex )
40       {
41           if ( nIndex >= getArea( ) )
42               return dData[ getArea( ) - 1 ];
43
44           return dData[ nIndex ];
45       }
46
47       double& get( const size_t x, const size_t y )
48       {
49           return dData[ ( y * nCols ) + x ];
50       }
51
52       double get( const size_t x, const size_t y ) const
53       {
54           return dData[ ( y * nCols ) + x ];
55       }
56
57       Matrix<nRows,nCols> add( const Matrix<nRows,nCols>& mOther )
58       {
59           Matrix<nRows,nCols> mReturn;
60
61           for ( size_t i=0; i < getArea( ); i++ )
62           {
63               mReturn.dData[i] = dData[i] + mOther.dData[i];
64           }
65
```

```cpp
66              return mReturn;
67      }
68
69      Matrix<nRows,nCols> operator+( const Matrix<nRows,nCols> & mOther )
70      {
71              return add( mOther );
72      }
73
74      template<short nCols2>
75      Matrix<nRows,nCols2> multiply( const Matrix<nCols,nCols2>& mOther )
76      {
77              /* Adapted from pseudocode on https://en.wikipedia.org/wiki/
                    Matrix_multiplication_algorithm */
78
79              Matrix<nRows,nCols2> mReturn;
80
81              for ( size_t i=0; i < nRows; i++ )
82              {
83                  for ( size_t j=0; j < nCols2; j++ )
84                  {
85                      double dSum = 0;
86
87                      for ( size_t k=0; k < nCols; k++ )
88                      {
89                          dSum += get( k, i ) * mOther.get( j, k );
90                      }
91
92                      mReturn.get( j, i ) = dSum;
93                  }
94              }
95
96              return mReturn;
97      }
98
99      template<short nCols2>
100     Matrix<nRows,nCols2> operator*( const Matrix<nCols,nCols2>& mOther )
101     {
102             return multiply( mOther );
103     }
104
105     friend std::ostream& operator<<( std::ostream& stream, const Matrix& mMatrix )
106     {
107             stream << "(" << nRows << "x" << nCols << ")[";
108
109             for ( size_t i=0; i < mMatrix.getArea( ); i++ )
110             {
111                 stream << ' ' << mMatrix.dData[i];
112
113                 if ( ( i + 1 ) % nCols == 0 && i != mMatrix.getArea( )-1 )
114                     stream << ',';
115                 else
116                     stream << ' ';
117             }
118
119             stream << "]";
120
121             return stream;
122     }
123 };
```

```
124
125  #endif
```

### 3.2.3  Compiler Output

Listing 16: ../lab/matrix/compilerout

```
1        matrix git:(master)        make CC=harper_cpp
2  harper_cpp  -std=c++14 main.cpp -o matrix.out
3  main.cpp***
```

### 3.2.4  Program Output

Listing 17: ../lab/matrix/progout

```
1        matrix git:(master)        ./matrix.out
2  Matrices:
3  a:     (3x2)[ 1   2, 3   4, 5   6 ]
4  b:     (2x3)[ 7   8   9, 10   11   12 ]
5  c:     (2x3)[ 14   15   16, 17   18   19 ]
6
7  a*b: (3x3)[ 27   30   33, 61   68   75, 95   106   117 ]
8  b+c: (2x3)[ 21   23   25, 27   29   31 ]
```

## 3.3  Sort Int Array

Name:         Nicolas Nytko                                    Course: CSC216

Activity:     Sort Int Array
Level:        3
Description:  C-3.18. Write a short recursive C++ function that will rearrange an array of int values so
              that all the even values appear before the odd values.

### 3.3.1  Compiler Environment

Listing 18: environment

```
1        tex git:(master)        pwd
2  /Users/nicolas/Git/portfolio1/tex
3        tex git:(master)        uname -a
4  Darwin Nicolass-MacBook-Pro.local 16.0.0 Darwin Kernel Version 16.0.0: Mon Aug 29
       17:56:20 PDT 2016; root:xnu-3789.1.32~3/RELEASE_X86_64 x86_64
5        tex git:(master)        clang --version
6  Apple LLVM version 8.0.0 (clang-800.0.38)
7  Target: x86_64-apple-darwin16.0.0
8  Thread model: posix
9  InstalledDir: /Library/Developer/CommandLineTools/usr/bin
10       ~ harper_cpp --version
11  This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:
12
13  Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include
       -dir=/usr/include/c++/4.2.1
14  Apple LLVM version 8.0.0 (clang-800.0.38)
15  Target: x86_64-apple-darwin16.1.0
16  Thread model: posix
17  InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

### 3.3.2 Source

Listing 19: ../lab/sortint/main.cpp

```cpp
1  #include <iostream>
2  #include <utility>
3
4  /* Recursively rearrange an array so that all even numbers
5   * will appear before odd ones.  Rearranged array will not
6   * be sorted and is not stable */
7
8  void sort( int* pArray, size_t nLength )
9  {
10     long nEven = -1, nOdd = -1;
11
12     for ( long i=0; i < static_cast<long>( nLength ) && ( nEven == -1 || nOdd == -1
          ); i++ )
13     {
14         if ( nEven == -1 && pArray[i] % 2 == 0 )
15             nEven = i;
16
17         if ( nOdd == -1 && pArray[i] % 2 == 1 )
18             nOdd = i;
19     }
20
21     if ( nEven != -1 && nOdd != -1 )
22     {
23         std::swap( pArray[nEven], pArray[nOdd] );
24     }
25
26     if ( nLength > 1 )
27     {
28         sort( pArray + 1, nLength - 1 );
29     }
30  }
31
32  template<typename T>
33  void printArray( T* pArray, size_t nLength )
34  {
35     for ( size_t i=0; i < nLength; i++ )
36     {
37         std::cout << pArray[i] << ' ';
38     }
39
40     std::cout << std::endl;
41  }
42
43  int main( )
44  {
45     const int LENGTH = 20;
46     int example[LENGTH] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
47                             11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
48
49     std::cout << "Array pre sorting:" << std::endl;
50     printArray( example, LENGTH );
51
52     sort( example, LENGTH );
53
54     std::cout << "Array post sorting:" << std::endl;
```

```
55        printArray( example , LENGTH );
56
57        return 0;
58 }
```

### 3.3.3  Compiler Output

Listing 20: ../lab/sortint/compilerout

```
1        sortint git :( master )        make CC=harper_cpp
2 harper_cpp   −std=c++14 main.cpp −o sort.out
3 main.cpp***
```

### 3.3.4  Program Output

Listing 21: ../lab/sortint/progout

```
1        sortint git :( master )        ./ sort.out
2 Array pre sorting:
3 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
4 Array post sorting:
5 2 4 6 8 10 12 14 16 18 20 11 3 13 7 15 1 17 9 19 5
```