C++ Exercises
Set 5

Author(s): Tjalling Otter, Emiel Krol

October 14, 2018

42

Listing 1: `main.ih`

```
#include <iostream>
#include <string>
#include "charcount/charcount.h"

using namespace std;

void showChar(CharCount charobject);
```

Listing 2: `main.cc`

```
#include "main.ih"

int main(int argc, char **argv)
{

  CharCount charObject(std::cin);

  showChar(charObject);
}
```

Listing 3: `showChar.cc`

```
// Programming in C/C++
// Week 4: Assignment 42
// Tjalling Otter & Emiel Krol

#include "charcount/charcount.h"
using namespace std;

void showChar(CharCount input)
{
  for (size_t index = 0; index < input.d_charObject.nChar; ++index)
  {
    if (input.getCount(index) != 0)
    {
      if (input.getChar(index) == ' ')
        cout << ' ' << "  :" << '\t';
      else if(input.getChar(index) == '\t')
        cout << "\\t" << ':' << '\t';
      else if(input.getChar(index) == '\n')
        cout << "\\n" << ':' << '\t';
      else if(isprint(input.getChar(index)))
        cout << "\"" << input.getChar(index) << "\"" << ':' << '\t';
      else
        cout << (int)input.getChar(index) + 128 << ':' << '\t';
    }
    cout << input.getCount(index) << '\n';
  }
}

//printing output
```

Listing 4: charcount/charcount.h

```
#ifndef INCLUDED_CHARCOUNT_
#define INCLUDED_CHARCOUNT_

#include <iostream>


class CharCount
{
  public:
    struct Char
    {
      char ch;       //the character
      size_t count = 0; //number of occurences
    };

    struct CharInfo
    {
      Char *ptr = new Char[1]; //field ptr pointing to struct char objects
      size_t nChar = 0; //nr char objects stored
    };

    CharInfo d_charObject;

    CharCount(std::istream& stream);
    Char *enlarge(Char *old, size_t oldsize, size_t newsize);
    char getChar(size_t index);
    void firstChar(char character);
    void existsCheck(char character, bool *exists);
    void newChar(char character);
    size_t getCount(size_t index);
    Char *charSorter(Char *array, size_t size);
    CharInfo *info();

};

inline char CharCount::getChar(size_t index)
{
  return d_charObject.ptr[index].ch;
}

inline size_t CharCount::getCount(size_t index)
{
  return d_charObject.ptr[index].count;
}


#endif
```

Listing 5: charcount/charcount.ih

```
#include "charcount.h"


using namespace std;
```

Listing 6: charcount/charSorter.cc

```
#include "charcount.ih"



CharCount::Char *CharCount::charSorter(Char *array, size_t size)
{
  for  (size_t idx = 0; idx < size; ++idx)
  {
    for (size_t index = 0 ; index < size - idx; ++index)
    {
      if  (array[index].ch > array[index + 1].ch)
        swap(array[index], array[index+1]);
```

```
    }
  }
  return array;
}

//At the first iteration the inner for loop puts the highest valued value
//on the highest index
//This inner for loop has to happen size times. Which is done with the outer
//for loop. After each iteration of the outer for loop, the latest highest value
//does not have to be checked again.
```

Listing 7: charcount/constructor.cc

```
#include "charcount.ih"
#include <iostream>
#include <stdlib.h>

using namespace std;

CharCount::CharCount(std::istream& stream)
{
  char character;
  size_t count = 0;

  while (stream.get(character))
  {
    count++;
    bool toCreateNewChar = true;
    bool exists = false;

    if (d_charObject.nChar == 0)
    {
      firstChar(character); //assigns the first char to Char[0]
      toCreateNewChar = false; //bool to indicate we dont need to increase
                               // the size of Char[]
    }
    else
      existsCheck(character, &exists); //checking if the char already exists
                                       //adding 1 to its counter if it does
                                       //and setting exists to true and
                                       //setting exists to false if it does not
                                       //exist.
    if (!exists && toCreateNewChar)
      newChar(character);              //increasing the size of Char[] and
                                       //assigning the char to ch and 1 to its
                                       //counter
  }
    d_charObject.ptr = charSorter(d_charObject.ptr, d_charObject.nChar - 1);
    //Sorting alphabetically
}
```

Listing 8: charcount/enlarge.cc

```
#include "charcount.ih"

CharCount::Char *CharCount::enlarge(Char *old, size_t oldsize, size_t newsize)
{
  Char *tmp = new Char[newsize];

  for (size_t idx = 0; idx != oldsize ; ++idx)
    tmp[idx] = old[idx];

  delete[] old;

  return tmp;
}
```

Listing 9: charcount/existsCheck.cc

```
#include "charcount.ih"
```

```
void CharCount::existsCheck(char character, bool *exists)
{
  for (size_t index = 0; index < d_charObject.nChar; ++index)
  {
    if  (d_charObject.ptr[index].ch == character)
    {
        d_charObject.ptr[index].count += 1;

        *exists = true;
    }
  }

}
```

Listing 10: charcount/firstChar.cc

```
#include "charcount.ih"

void CharCount::firstChar(char character)
{
    d_charObject.ptr[0].ch = character;
    d_charObject.nChar = 1;
    d_charObject.ptr[0].count = 1;
}
```

Listing 11: charcount/info.cc

```
#include "charcount.ih"

CharCount::CharInfo *CharCount::info()
{
  return &d_charObject;

}
```

Listing 12: charcount/newChar.cc

```
#include "charcount.ih"

void CharCount::newChar(char character)
{
    d_charObject.ptr = enlarge(d_charObject.ptr, d_charObject.nChar,
       d_charObject.nChar + 1);

    d_charObject.ptr[d_charObject.nChar].count = 1;

    d_charObject.ptr[d_charObject.nChar].ch = character;

    d_charObject.nChar += 1;
}
```


Listing 13: main.ih

```
// Main: internal header file

#include "matrixFunctions.h"

using namespace std;
```

Listing 14: main.cc

```
// Main file
```

```cpp
#include "main.ih"

int main()
{
  int square[DIM][DIM]; // Declare square 2D array

  int (*row)[DIM] = square;  // Initialise row as pointing to rows of 2D array

  inv_identity(row);  // Pass row to function
  // printArray(square, DIM); // Only for testing purposes
}
```

Listing 15: matrixFunctions.h

```cpp
// Matrix functions: header file

#ifndef INCLUDED_MATRIXFUNCTIONS_
#define INCLUDED_MATRIXFUNCTIONS_

#include <cstddef>

enum FIXEDVARS
{
  DIM = 10
};

void inv_identity(int entryRow[][DIM]);
void allOnes(int entryRow[][DIM]);
void diagZeroes(int entryRow[][DIM]);

void printArray(int const square[][DIM]);  // Testing purposes

#endif
```

Listing 16: inv_identity.cc

```cpp
// Matrix function: make array into inverted identity matrix

#include "matrixFunctions.h"

void inv_identity(int entryRow[][DIM])
{
  allOnes(entryRow);     // Make all entries ones
  diagZeroes(entryRow); // Make diagonal zeroes
};
```

Listing 17: allOnes.cc

```cpp
// Matrix function: make 2D array into matrix of ones

#include "matrixFunctions.h"

void allOnes(int entryRow[][DIM])
{
  for (int (*row)[DIM] = entryRow; row != entryRow + DIM; ++row)
  {
    for (int *column = *row, *end = *row + DIM; column != end; ++column)
      (*column) = 1;
  }
};
// Loop through all the rows, then within those rows the columns (now individual
// elements) and set them all to one.
```

Listing 18: diagZeroes.cc

```cpp
// Matrix function: make diagnonal zeroes

#include "matrixFunctions.h"
```

```
void diagZeroes(int entryRow[][DIM])
{
  for (int *entry = *entryRow, index = 0; index != DIM; ++index, entry += DIM +
     1)
    (*entry) = 0;
};
// Sets every eleventh (dimension + 1) element to zero
```