```
************************
**                    **
**     PROBLEM #1     **
**                    **
************************

/*
 *  Project:    HW #1, Problem # 1
 *  Author:     Matthew Springer
 *  Date:       January 30, 2017
 *  Purpose:    Read two integers from stdin and performs a
number of operations on them
 */
#include <iostream>
#include <string>
#include <cmath>
#include <algorithm>
#include <bitset>

using namespace std;

/*
 *  Function: PrintDataSizes
 *
 *  Purpose:  Prints out the number of bytes used to store
various data types
 */
void PrintDataSizes() {
  cout << "A   bool requires 1 bytes of memory" << endl;
  cout << "A   char requires 1 bytes of memory" << endl;
  cout << "A    int requires 4 bytes of memory" << endl;
  cout << "A  float requires 4 bytes of memory" << endl;
  cout << "A double requires 8 bytes of memory" << endl;
}

/*
 *  Function: Power
 *
 *  @param  a int
 *  @param  b int
 *  @return a^b
 */
int Power(int a, int b) {
  return pow(a, b);
}
```

```
/*
 *  Function: Maximum
 *
 *  @param  a int
 *  @param  b int
 *  @return max(a, b)
 */
int Maximum(int a, int b) {
  return max(a, b);
}

/*
 *  Function: PrintInts
 *
 *  @param  a int
 *  @param  b int
 *
 *  Purpose: print two given integers in decimal, binary, hex,
and octal form
 */
void PrintInts(int a, int b) {
  bitset<8> a_bin(a);
  bitset<8> b_bin(b);
  cout << "Decimal: " << a << ", " << b << endl;
  cout << "Binary: " << a_bin << ", " << b_bin << endl;
  cout << "Octal: " << oct << a << ", " << oct << b << endl;
  cout << "Hexadecimal: " << hex << a << ", " << hex << b <<
endl;
}

int main() {
  int x, y;
  PrintDataSizes();
  cout << "Please enter the first integer: ";
  cin >> x;
  cout << "Please enter the second integer: ";
  cin >> y;
  cout << x << "^" << y << " = " << Power(x, y) << endl;
  cout << "Max(" << x << ", " << y << ") = " << Maximum(x, y) <<
endl;
  PrintInts(x, y);
}
```

** Console Output **

```
Matts-MacBook-Pro:hw01 mspringer$ g++ problem1.cpp -o problem1
Matts-MacBook-Pro:hw01 mspringer$ ./problem1
A   bool requires 1 bytes of memory
A   char requires 1 bytes of memory
A    int requires 4 bytes of memory
A  float requires 4 bytes of memory
A double requires 8 bytes of memory
Please enter the first integer: 16
Please enter the second integer: 3
16^3 = 4096
Max(16, 3) = 16
Decimal: 16, 3
Binary: 00010000, 00000011
Octal: 20, 3
Hexadecimal: 10, 3
```

```
************************
**                    **
**     PROBLEM #2     **
**                    **
************************


/*
 *  Project:    HW #1, Problem # 2
 *  Author:     Matthew Springer
 *  Date:       January 30, 2017
 *  Purpose:    Write a program that produces the transpose of a
3x3 matrix
 */

#include <iostream>

using namespace std;

/*
 *  Function:   NewMatrix3x3
 *  Input:      None
 *  Output:     int ** Matrix3x3
 *  Purpose:    Generates an empty 3x3 matrix of integers
 */
int ** NewMatrix3x3() {
  int ** Matrix3x3 = new int*[3];
  Matrix3x3[0] = new int[3];
  Matrix3x3[1] = new int[3];
  Matrix3x3[2] = new int[3];
  return Matrix3x3;
}

/*
 *  Function:   InitMatrix
 *  Input:      int** matrix, int rows (number of rows), int
columns (number of columns)
 *  Output:     None
 *  Purpose:    Sets a matrix's values to numerically-ordered
integers (row-wise)
 */
void  InitMatrix(int** matrix, int rows, int columns) {
  int counter = 0;
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
      matrix[j][i] = counter++;
    }
  }
}
```

```
/*
 *  Function:   NewMatrix
 *  Input:      int rows (number of rows), int columns (number of
columns)
 *  Output:     int ** Matrix (new integer matrix of size rows x
columns)
 *  Purpose:    Generates an empty matrix of size rows x columns
 */
int ** NewMatrix(int rows, int columns) {
  int ** MatrixRxC = new int*[columns];
  for (int i = 0; i < columns; i++) {
    MatrixRxC[i] = new int[rows];
  }
  return MatrixRxC;
}

/*
 *  Function:   DuplicateMatrix
 *  Input:      int ** MatrixIn, int rows (number of rows), int
columns (number of columns)
 *  Output:     int ** MatrixOut, (new integer matrix of size
rows x columns)
 *  Purpose:    Generates an empty matrix of size rows x columns
 */
int ** DuplicateMatrix(int ** matrix, int rows, int columns) {
  int ** DupMat = new int*[columns];
  for (int i = 0; i < columns; i++) {
    DupMat[i] = new int[rows];
    for (int j = 0; j < rows; j++) {
      DupMat[i][j] = matrix[i][j];
    }
  }
  return DupMat;
}

/*
 *  Function:   DeleteMatrix
 *  Input:      int** matrix, int columns (number of columns)
 *  Output:     None
 *  Purpose:    Deallocates memory for dynamically-generated
matrix with a given number of columns
 */
void DeleteMatrix(int ** matrix, int columns) {
  for (int i = 0; i < columns; i++) {
    delete [] matrix[i];
  }
  delete [] matrix;
}
```

```
/*
 *  Function:   PrintMatrix
 *  Input:      int** matrix, int rows (number of rows), int
columns (number of columns)
 *  Output:     None
 *  Purpose:    Prints a given matrix of size rows x columns
 */
void PrintMatrix(int** matrix, int rows, int columns) {
  for (int i = 0; i < rows; i++) {
    cout << endl << "\t|\t";
    for (int j = 0; j < columns; j++) {
      cout << matrix[j][i] << "\t";
    }
    cout << "|" << endl;
  }
}

/*
 *  Function:   IndexTranspose
 *  Input:      int** matrix, int rows (number of rows), int
columns (number of columns)
 *  Output:     None
 *  Purpose:    Mutates a given Matrix to its transpose, using
array indices
 */
void IndexTranspose(int ** matrix, int rows, int columns) {
  int ** copyMat = DuplicateMatrix(matrix, rows, columns);
  DeleteMatrix(matrix, columns);
  matrix = NewMatrix(columns, rows);
  //int ** transpose = NewMatrix(columns, rows);
  for (int i = 0; i < columns; i++) {
    for (int j = 0; j < rows; j++) {
      matrix[j][i] = copyMat[i][j];
    }
  }
}
```

```
/*
 *  Function:   PointerTranspose
 *  Input:      int** matrix, int rows (number of rows), int
columns (number of columns)
 *  Output:     None
 *  Purpose:    Mutates a given Matrix to its transpose, using
pointers
 */
void PointerTranspose(int ** matrix, int rows, int columns) {
  int ** copyMat = DuplicateMatrix(matrix, rows, columns);
  DeleteMatrix(matrix, columns);
  matrix = NewMatrix(columns, rows);
  //int ** transpose = NewMatrix(columns, rows);
  for (int i = 0; i < columns; i++) {
    for (int j = 0; j < rows; j++) {
      *(*(matrix+j) + i) = *(*(copyMat+i) + j);
    }
  }
}

int main() {
  int rows, cols;
  rows = 3;
  cols = 3;
  int** mat = NewMatrix(rows,cols);
  int ** transpose;
  InitMatrix(mat, rows, cols);
  cout << "Initial Matrix: " << endl;
  PrintMatrix(mat, rows, cols);
  IndexTranspose(mat, rows, cols);
  cout << "Transposed Matrix using indices: " << endl;
  PrintMatrix(mat, cols, rows);
  PointerTranspose(mat, rows, cols);
  cout << "Transpose of Transposed Matrix using pointers: " <<
endl;
  PrintMatrix(mat, cols, rows);
  //PrintMatrix(transpose, cols, rows);

  return 0;
}
```

```
** Console Output **

Matts-MacBook-Pro:hw01 mspringer$ g++ problem2.cpp -o problem2
Matts-MacBook-Pro:hw01 mspringer$ ./problem2
Initial Matrix:

      |     0     1     2     |

      |     3     4     5     |

      |     6     7     8     |
Transposed Matrix using indices:

      |     0     3     6     |

      |     1     4     7     |

      |     2     5     8     |
Transpose of Transposed Matrix using pointers:

      |     0     1     2     |

      |     3     4     5     |

      |     6     7     8     |
Matts-MacBook-Pro:hw01 mspringer$
```

```
************************
**                    **
**     PROBLEM #3     **
**                    **
************************

/*
 *  Project:    HW #1, Problem # 3
 *  Author:     Matthew Springer
 *  Date:       January 30, 2017
 *  Purpose:    Write a program that produces an array of
CarRecords from an
 *              input file and performs operations on the array
 */

// Libraries
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

/*
 *  Struct Name:  CarRecord
 *
 *  @param  make  the make of the car
 *  @param  model the model of the car
 *  @param  year  the year of the car
 *  @param  color the color of the car
 */
struct CarRecord {
  string make;
  string model;
  int year;
  string color;
};

/*
 *  Function: LinesInFile
 *
 *  @param  file_name the name of the file to read from
 *  @return int       the number of lines in the file
 */
int LinesInFile(string file_name) {
  ifstream myfile;
  string line;
  int numLines = 0;
  myfile.open(file_name);
  while (getline(myfile, line))
    numLines++;
  return numLines;
}
```

```
/*
 *  Function: PopulateRecordFromLine
 *
 *  @param  record  the CarRecord to populate
 *  @param  line    the line to populate the record from
 *  @return None
 */
void PopulateRecordFromLine(CarRecord * record, string line) {
}

/*
 *  Function: Insert_Array
 *
 *  @param  array     the array of CarRecords to populate
 *  @param  lines    the number of lines in the file
 *  @param  inputFile the name of the file to read from
 *  @return None
 */
void Insert_Array(CarRecord * array, int lines, string inputFile)
{
  ifstream myfile;
  string line;
  string make;
  string model;
  string color;
  string yearStr;
  int yearInt;
  myfile.open(inputFile);
  int lineCounter = 0;
  while (!myfile.eof()) {
    for (int i = 0; i < lines; i++) {
      if(getline(myfile, make, ',')){
        //cout << make << " ";
        array[i].make = make;
      }
      if (getline(myfile, model, ',')) {
        //cout << model << " ";
        array[i].model = model;
      }
      if (getline(myfile, yearStr, ',')) {
        //cout << yearStr << " ";
        yearInt = stoi(yearStr);
        array[i].year = yearInt;
      }
      if (getline(myfile, color)) {
        //cout << color << endl;
        array[i].color = color;
      }
    }
  }
}
```

```
/*
 *  Function: Sort_Cars_By_Year
 *
 *  @param  array     the array of CarRecords to sort (by year)
 *  @param  legnth    the length of the array
 *  @return None
 */
void Sort_Cars_By_Year(CarRecord * array, int length) {
  int lowestIndex;
  CarRecord * placeholderRecord = new CarRecord;
  for (int i = 0; i < length - 1; i++) {
    lowestIndex = i;
    for (int j = i+1; j < length; j++) {
      if (array[j].year < array[lowestIndex].year) {
        lowestIndex = j;
      }
    }
    if (lowestIndex > i) {
      placeholderRecord->make = array[i].make;
      placeholderRecord->model = array[i].model;
      placeholderRecord->year = array[i].year;
      placeholderRecord->color = array[i].color;

      array[i].make = array[lowestIndex].make;
      array[i].model = array[lowestIndex].model;
      array[i].year = array[lowestIndex].year;
      array[i].color = array[lowestIndex].color;

      array[lowestIndex].make = placeholderRecord->make;
      array[lowestIndex].model = placeholderRecord->model;
      array[lowestIndex].year = placeholderRecord->year;
      array[lowestIndex].color = placeholderRecord->color;
    }
  }
}
```

```
/*
```

```
 *  Function: Print_Duplicates
 *
 *  @param  array     the array of CarRecords to scan for
duplicates to print
 *  @param  legnth    the length of the array
 *  @return None
 */
void Print_Duplicates(CarRecord * array, int length) {
  bool hasDupes = false;
  bool * duplicates = new bool[length];
  for (int i = 0; i < length - 1; i++) {
    for (int j = i+1; j < length; j++) {
      if (array[i].make == array[j].make &&
          array[i].model == array[j].model &&
          array[i].year == array[j].year &&
          array[i].color == array[j].color) {
        duplicates[i] = true;
        duplicates[j] = true;
        hasDupes = true;
      }
    }
  }
  if (hasDupes) {
    cout << "Duplicate Entries: " << endl << endl;
    for (int i = 0; i < length - 1; i++) {
      if (duplicates[i]) {
        cout << i+1 << ".\t";
        cout << array[i].make << " ";
        cout << array[i].model << " ";
        cout << array[i].year << " ";
        cout << array[i].color << endl;
      }
    }
    cout << endl;
  }
  else cout << "No duplicate entries" << endl << endl;
}
```

```
/*
 *  Function: Print_Cars_Array
 *
 *  @param  array     the array of CarRecords to print
 *  @param  legnth    the length of the array
 *  @return None
 */
void Print_Cars_Array(CarRecord * array, int length) {
  cout << "Car Records:" << endl << endl;
  string make;
  string model;
  string color;
  int year;
  string space = " ";
  for (int i = 0; i < length; i++) {
    int recordNum = i+1;
    make = array[i].make;
    model = array[i].model;
    year = array[i].year;
    color = array[i].color;

    cout << recordNum << ".\t";
    cout << make << space;
    cout << model << space;
    cout << year << space;
    cout << color << endl;
  }
  cout << endl;
}

/*
 *  Function: Print_Main_Menu
 *  MENU - Select an option:
 1. Print the cars array
 2. Insert car records into a sorted array
 3. Sort cars by year
 4. Print duplicates
 5. Exit
 */
void Print_Main_Menu() {
  cout << "MAIN MENU:" << endl << endl;
  cout << "1. Print the cars array" << endl;
  cout << "2. Insert car records into an array" << endl;
  cout << "3. Sort the array by year" << endl;
  cout << "4. Print duplicates" << endl;
  cout << "5. Exit" << endl << endl;
  cout << "Please enter an option from the menu above: ";
}
```

```
int main() {
  string file_name = "CarRecords.txt";
  int numRecords = LinesInFile(file_name);
  CarRecord * recordsArray = new CarRecord[numRecords];
  bool executing = true;
  while (executing) {
    char input;
    Print_Main_Menu();
    cin >> input;
    cout << endl;
    switch (input) {
      case '1':
        Print_Cars_Array(recordsArray, numRecords);
        break;
      case '2':
        Insert_Array(recordsArray, numRecords, file_name);
        break;
      case '3':
        Sort_Cars_By_Year(recordsArray, numRecords);
        break;
      case '4':
        Print_Duplicates(recordsArray, numRecords);
        break;
      case '5':
        executing = false;
        break;
      default:
        cout << "You have entered an invalid input.  Please enter
the number that corresponds with the command you wish to execute.
Re-displaying main menu..." << endl;
        cin.clear();
        cin.ignore(10000,'\n');
        break;
    }
  }
  return 0;
}
```

```
** CONSOLE OUTPUT **

Matts-MacBook-Pro:hw01 mspringer$ ./problem3
MAIN MENU:

1. Print the cars array
2. Insert car records into a sorted array
3. Sort cars by year
4. Print duplicates
5. Exit

Please enter an option from the menu above: 1

Car Records:

1.      0
2.      0
3.      0
4.      0
5.      0
6.      0
7.      0
8.      0
9.      0
10.     0

MAIN MENU:

1. Print the cars array
2. Insert car records into a sorted array
3. Sort cars by year
4. Print duplicates
5. Exit

Please enter an option from the menu above: 2

MAIN MENU:

1. Print the cars array
2. Insert car records into a sorted array
3. Sort cars by year
4. Print duplicates
5. Exit

Please enter an option from the menu above: 1
```

```
Car Records:

1.    Subaru Outback 2016 green
2.    Toyota Corolla 2006 white
3.    Dodge Neon 1993 pink
4.    Ford Fusion 2013 yellow
5.    Honda Fit 2015 blue
6.    Ford Expedition 2009 silver
7.    Toyota Corolla 2006 white
8.    Ford Fusion 2013 yellow
9.    Jeep Cherokee 1999 red
10.   Mazda Protoge 1996 gold

MAIN MENU:

1. Print the cars array
2. Insert car records into a sorted array
3. Sort cars by year
4. Print duplicates
5. Exit

Please enter an option from the menu above: 3

MAIN MENU:

1. Print the cars array
2. Insert car records into a sorted array
3. Sort cars by year
4. Print duplicates
5. Exit

Please enter an option from the menu above: 1

Car Records:

1.    Dodge Neon 1993 pink
2.    Mazda Protoge 1996 gold
3.    Jeep Cherokee 1999 red
4.    Toyota Corolla 2006 white
5.    Toyota Corolla 2006 white
6.    Ford Expedition 2009 silver
7.    Ford Fusion 2013 yellow
8.    Ford Fusion 2013 yellow
9.    Honda Fit 2015 blue
10.   Subaru Outback 2016 green
```

```
MAIN MENU:

1. Print the cars array
2. Insert car records into a sorted array
3. Sort cars by year
4. Print duplicates
5. Exit

Please enter an option from the menu above: 4

Duplicate Entries:

4.    Toyota Corolla 2006 white
5.    Toyota Corolla 2006 white
7.    Ford Fusion 2013 yellow
8.    Ford Fusion 2013 yellow

MAIN MENU:

1. Print the cars array
2. Insert car records into a sorted array
3. Sort cars by year
4. Print duplicates
5. Exit

Please enter an option from the menu above: 5

Matts-MacBook-Pro:hw01 mspringer$
```