```
///////////////////////////////////////////////////////////////////////////////
//
// Author:           Miklos Moreno
// Email:            miklosam1999@gmail.com
// Label:            P01
// Title:            Program 1 - Resizing the Stack
// Course:           3013
// Semester:         Spring 2022
//
// Description:
//       Array based stack with resize methods based on input from file.
//
/////////////////////////////////////////////////////////////////////////////////
#include <iostream>
#include <fstream>
#include <cmath>  // didnt like how typecast rounds down
                  // used 'round()' in grow and shrink methods

using namespace std;

ifstream infile;
ofstream ofile;


/**
 * ArrayStack
 *
 * Description:
 *      Array based stack
 *
 * Public Methods:
 *      - ArrayStack()
 *      - ArrayStack(int)
 *      - bool Empty()
 *      - bool Full()
 *      - int Peek()
 *      - int Pop()
 *      - void Print()
 *      - bool Push(int)
 *      - void ContainerGrow()
 *      - void ContainerShrink()
 *      - void CheckResize()
 *      - int getSize()
 *      - int getTimesResized()
 *      - int getMaxSize()
 *      - int getTop()
 *      - double getGrowThresh()
 *      - double getShrinkThresh()
 *      - double getShrinkRatio()
 *      - double getGrowRatio()
 *      - void setGrowThresh(double)
 *      - void setGrowRatio(double)
 *      - void setShrinkRatio(double)
 *      - void setShrinkThresh(double)
 *
 * Usage:
 *      - See main program
 *
 */
class ArrayStack {
    int *A;               // pointer to array of int's
    int size;             // current max stack size
    int top;              // top of stack
    int resizeCount;      // times stack has been resized
    int maxSize;          // max size stack reaches
    double growThresh;    // threshold for growing stack
    double shrinkThresh; // threshold for shrinking stac
    double growRatio;     // ratio for growing stack
    double shrinkRatio;   // ratio for shrinking stack

public:
    /**
  * ArrayStack
  *
  * Description:
  *      Array based stack with resize methods
  *
  * Public Methods:
  *      -
  *
  * Returns:
  *      - NULL
  */
    ArrayStack()
    {
        size = 10;
        A = new int[size];
        top = -1;
        maxSize = size;
        resizeCount = 0;
    }

    /**
  * ArrayStack
  *
  * Description:
  *      Constructor size param
  *
  * Params:
  *      - int size
  *
  * Returns:
  *      - NULL
  */
    ArrayStack(int s)
    {
        size = s;
        A = new int[s];
        top = -1;
    }

    /**
  * Public bool: Empty
  *
  * Description:
  *      Stack empty?
  *
  * Params:
  *      NULL
  *
  * Returns:
  *      [bool] true = empty
  */
    bool Empty(){return (top <= -1);}

    /**
  * Public bool: Full
  *
  * Description:
  *      Stack full?
  *
  * Params:
  *      NULL
  *
  * Returns:
  *      [bool] true = full
  */
    bool Full() {return (top >= size - 1);}

    /**
  * Public int: Peek
  *
  * Description:
  *      Returns top value without altering the stack
  *
  * Params:
  *      NULL
  *
  * Returns:
  *      [int] top value if any
  */
    int Peek()
    {
        if (!Empty()) {
            return A[top];
        }

        return -99; // some sentinel value
                    // not a good solution
    }
    /**
  * Public int: Pop
  *
  * Description:
  *      Returns top value and removes it from stack
  *
  * Params:
  *      NULL
  *
  * Returns:
  *      [int] top value if any
  */
    int Pop()
    {
        if (!Empty()) {
            return A[top--];
        }

        return -99; // some sentinel value
                    // not a good solution
    }

    /**
  * Public void: Print
  *
  * Description:
  *      Prints stack to standard out
  *
  * Params:
  *      NULL
  *
  * Returns:
  *      NULL
  */
    void Print()
    {
        for (int i = 0; i <= top; i++) {
            cout << A[i] << " ";
        }
        cout << endl;
    }

    /**
  * Public bool: Push
  *
  * Description:
  *      Adds an item to top of stack
  *
  * Params:
  *      [int] : item to be added
  *
  * Returns:
  *      [bool] : success = true
```

```
    */
      bool Push(int x)
      {
          A[++top] = x;
          CheckResize();
          return true;
      }

      /**
    * Public void: ContainerGrow
    *
    * Description:
    *      Resizes the container for the stack by mult.
    *      size by growth ratio
    * Params:
    *      NULL
    *
    * Returns:
    *      NULL
    */
      void ContainerGrow()
      {
          int newSize = size * 1.50;  // set new size based on growth ratio
          int *B = new int[newSize];                 // allocate new memory

          for (int i = 0; i < top; i++)
          {
              B[i] = A[i];        // copy values to new array
          }

          delete[] A; // delete old array

          size = newSize; // save new size

          A = B; // reset array pointer

          if (maxSize > newSize)
          {
              maxSize = size;
          }
      }

      /**
    * Public void: ContainerShrink
    *
    * Description:
    *      Resizes the container for the stack by mult.
    *      size by shrink ratio.
    *
    * Params:
    *      NULL
    *
    * Returns:
    *      NULL
    */
      void ContainerShrink()
      {
          int newSize = size / 2;
          if (newSize < 10)
          {
              newSize = 10;
          }

          int *B = new int[newSize];

          for (int i = 0; i < top; i++)
          {
              B[i] = A[i];        // copy values to new array
          }

          delete[] A;

          size = newSize;

          A = B;
      }

      /**
       * Public void: CheckResize
       *
       * Description:
       *      Checks size of stacks and determines
       *      when to run grow or shrink method.
       *
       * Params:
       *      NULL
       *
       * Returns:
       *      NULL
       */
      void CheckResize()
      {
              if (Full())
          {
              ContainerGrow();                  // Call this function to shrink
              resizeCount++;                    // increments times resized
          }
          else if (top < (size / 2) && size > 10)
          {
              ContainerShrink();                // Call this function to grow
              resizeCount++;                    // increments times resized
          }
      }

      /**
       * Public int: getSize
       *
       * Description:
```

```
     *      returns size of stack
     *
     * Params:
     *      NULL
     *
     * Returns:
     *      int size
     */
    int getSize(){return size;}

    /**
     * Public int: getresizeCount
     *
     * Description:
     *      returns times of resize methods called
     *
     * Params:
     *      NULL
     *
     * Returns:
     *      int resizeCount
     */
    int getResizeCount(){return resizeCount;}

    /**
     * Public int: getMaxSize
     *
     * Description:
     *      returns max size of stack
     *
     * Params:
     *      NULL
     *
     * Returns:
     *      int maxSize
     */
    int getMaxSize(){return maxSize;}

    /**
     * Public int: getTop
     *
     * Description:
     *      returns top of stack
     *
     * Params:
     *      NULL
     *
     * Returns:
     *      int top
     */
    int getTop(){return top;}

  /**
   * Public Double: getGrowThresh
```

```
    *
    * Description:
    *       returns grow threshold
    *
    * Params:
    *       NULL
    *
    * Returns:
    *       double grow threshold
    */
     double getGrowThresh(){return growThresh;}

   /**
    * Public void: setGrowThresh
    *
    * Description:
    *       sets the grow threshold
    *
    * Params:
    *       NULL
    *
    * Returns:
    *        NULL
    */
     void setGrowThresh(double x){growThresh = x;}

 /**
  * Public double: getShrinkThresh
  *
  * Description:
  *       returns shrink threshold
  *
  * Params:
  *       NULL
  *
  * Returns:
  *       double shrink threshold
  */
     double getShrinkThresh(){return shrinkThresh;}

 /**
  * Public void setShrinkThresh
  *
  * Description:
  *         sets the shrink threshold
  *
  * Params:
  *       double x
  *
  * Returns:
  *       NULL
  */
     void setShrinkThresh(double x){shrinkThresh = x;}
```

```
/**
 * Public double: getGrowRatio
 *
 * Description:
 *      returns grow ratio
 *
 * Params:
 *      NULL
 *
 * Returns:
 *      double grow ratio
 */
    double getGrowRatio(){return growRatio;}

/**
 * Public void: setGrowRatio
 *
 * Description:
 *        sets grow ratio
 *
 * Params:
 *      double x
 *
 * Returns:
 *      NULL
 */
    void setGrowRatio(double x){growRatio = x;}

/**
 * Public double getShrinkRatio
 *
 * Description:
 *      returns shrink ratio
 *
 * Params:
 *      NULL
 *
 * Params:
 *      double shrink ratio
 */
    double getShrinkRatio(){return shrinkRatio;}

/**
 * Public void: setShrinkRatio
 *
 * Description:
 *      sets shrink ratio
 *
 * Params:
 *      double x
 *
 * Returns:
 *      NULL
 */
```

```
    void setShrinkRatio(double x){shrinkRatio = x;}

};

void openFiles(ifstream& infile, ofstream& outfile)
{
    char inFileName[40];
    char outFileName[40];

    cout << "Enter the input file name: ";    // Prompt the User
    cin >> inFileName;

    infile.open(inFileName);                  // open input file
    cout << "Enter the output file name: ";
    cin >> outFileName;

    outfile.open(outFileName);                // Open output file.
}

// MAIN DRIVER
// Simple Array Based Stack Usage:
int main() {

  ArrayStack stack;
  int commandCount;
  int input;

  openFiles(infile, ofile);



  while(!infile.eof())
  {
    infile >> input;
    if (input % 2 == 0)
    {
      stack.Push(input);
    }else
    {
      stack.Pop();
    }
    commandCount++;
  }

  ofile << string(50, '#') << endl;
  ofile <<"Program 1 - Resizing the Stack"<< endl;
  ofile <<"CMPS 3013"<< endl;
  ofile <<"Miklos Moreno"<< endl << endl;
  ofile <<"Config Params:"<< endl;
  ofile <<"  Full Threshold: "<< stack.getGrowThresh() << endl;
  ofile <<"  Shrink Threshold: "<< stack.getShrinkThresh() << endl;
  ofile <<"  Grow Ratio: "<< stack.getGrowRatio() << endl;
  ofile <<"  Shrink Ratio: "<< stack.getShrinkRatio() << endl << endl;
  ofile << "Processed "<< commandCount << " commands." << endl << endl;
```

```
    ofile << "Max Stack Size: "<< stack.getMaxSize() << endl;
    ofile << "End Stack Size: "<< stack.getSize() << endl;
    ofile << "Stack Resized: "<< stack.getResizeCount()
          << " Times" << endl;
    ofile << string(50, '#');

    return 0;
  }
```