Final Exam
CS253 Fall 2014

Code Packet.
Do not put answers on this packet
(it will NOT be graded)

Instructions:

There are a total of 4 files in this code packet. There are three template files (Image.tpp, ImageIterator.tpp, and Window.tpp) and one code file (main.cpp). The template files are exactly the same as the ones given you as part of the practice final. The main.cpp file is different. Note also that (as before) Image.tpp is spread over 4 pages, Window.tpp has 2 pages, and main.cpp has 2 pages. Also, the code is printed double sided.

All of the code in this packet compiles without errors. What it does is up for you to determine ☺

```
/* FILE main.cpp */

#include <Image.tpp>
#include <Window.tpp>

#include<iostream>
using std::cout;
using std::endl;
#include<sstream>
using std::istringstream;

int main(int argc, char* argv[])
{
  if (argc != 2) return -1;

  istringstream istr(argv[1]);
  int index;
  istr >> index;
  if (istr.fail()) return -2;

  switch(index) {
  case 0: {
    Image<int> img(3, 5, 0);
    Window<int> win = img.GetWindow(0, 4, 2, 2);

    for(int r=0; r < 2; r++) {
      for(int c=0; c < 2; c++) {
        win(r,c) = 1;
      }
    }

    Image<int>::iterator end = img.end();
    for(Image<int>::iterator iter = img.begin();
        iter != end;
        ++iter) {
      cout << *iter << " ";
    }
    cout << endl;
    break;
  }
```

```
  case 1: {
    Image<int> img(3, 5, 0);
    Window<int> win = img.GetWindow(2, 4, 2, 2);

    for(int r=0; r < 2; r++) {
      for(int c=0; c < 2; c++) {
        win(r,c) = 1;
      }
    }

    Image<int>::iterator end = img.end();
    for(Image<int>::iterator iter = img.begin();
        iter != end;
        ++iter) {
      cout << *iter << " ";
    }
    cout << endl;
    break;
  }
  case 2: {
    Image<int> img;
    Window<int> win = img.GetWindow(0, 0, 2, 2);

    for(int r=0; r < 2; r++) {
      for(int c=0; c < 2; c++) {
        win(r,c) = 1;
      }
    }

    Image<int>::iterator end = img.end();
    for(Image<int>::iterator iter = img.begin();
        iter != end;
        ++iter) {
      cout << *iter << " ";
    }
    cout << endl;
  }
  }
}
```

```
/* File Image.tpp */
#ifndef IMAGE_TPP_DEFINED
#define IMAGE_TPP_DEFINED
#include<ImageIterator.tpp>
#include<vector>
using std::vector;

template<typename PIXEL>
class Window;

template<typename PIXEL>
class Image {
public:
  typedef ImageIterator<PIXEL> iterator;

  Image() : rows(0), cols(0) {}
  Image(unsigned int row_count, unsigned int col_count)
    : data(NULL) { AllocateMemory(row_count, col_count); }
  Image(unsigned int row_count, unsigned int col_count,
        PIXEL init_value) : data(NULL)
  { AllocateMemory(row_count, col_count); Set(init_value); }
  Image(const Image<PIXEL>& src) : data(NULL)
  { AllocateMemory(src.rows, src.cols); Copy(src); }
  virtual ~Image() { DeAllocateMemory(); }

  inline unsigned int Rows() const {return rows;}
  inline unsigned int Cols() const {return cols;}
  inline unsigned int Size() const {return size;}
  virtual unsigned int EndOfRowIncrement() const {return 1;}

  PIXEL& operator() (unsigned int row, unsigned int col)
  {return row_ptrs[row][col];}
  const PIXEL& operator() (unsigned int row, unsigned int col) const
  {return row_ptrs[row][col];}
  bool Inside(unsigned int row, unsigned int col) const
  {return ((row < rows) && (col < cols));}
  PIXEL& at(unsigned int row, unsigned int col)
  { if (Inside(row, col)) return row_ptrs[row][col];
    else throw std::exception();}
  const PIXEL& at(unsigned int row, unsigned int col) const
  {if (Inside(row, col)) return row_ptrs[row][col];
    else throw std::exception();}
```

```
  /* Continuation (Page #2) of File Image.tpp */

  bool empty() const {return (size == 0);}
  iterator begin()
  { return ImageIterator<PIXEL>(data, cols, EndOfRowIncrement()); }
  iterator end()
  { return ImageIterator<PIXEL>(data+size, cols, EndOfRowIncrement()); }
  Window<PIXEL> GetWindow(unsigned int start_row,
                          unsigned int start_col,
                          unsigned int height,
                          unsigned int width);

  void Set(PIXEL value);

protected:
  void AllocateMemory(unsigned int row_count, unsigned int col_count);
  void DeAllocateMemory();

  template<typename OTHER_PIXEL>
  void Copy(const Image<OTHER_PIXEL>& source_image);

  int rows;
  int cols;
  int size;
  PIXEL* data;
  PIXEL** row_ptrs;
};

#include <Window.tpp>
```

```
/* Continuation (Page #3) of Image.tpp */


template<typename PIXEL>
void Image<PIXEL>::AllocateMemory(unsigned int row_count, unsigned int
col_count)
{
  if (data != NULL) DeAllocateMemory();

  rows = row_count;
  cols = col_count;
  size = rows * cols;

  data = new PIXEL[size];
  row_ptrs = new PIXEL*[rows];
  PIXEL* data_iter = data;
  PIXEL** end = row_ptrs + rows;
  for(PIXEL** row_iter = row_ptrs;
      row_iter != end;
      row_iter++, data_iter += cols) {
    (*row_iter) = data_iter;
  }
}

template<typename PIXEL>
void Image<PIXEL>::DeAllocateMemory()
{
  rows = 0;
  cols = 0;
  delete [] data;
  data = NULL;
  delete [] row_ptrs;
}

template<typename PIXEL>
void Image<PIXEL>::Set(PIXEL value)
{
  PIXEL* end = data + size;
  for(PIXEL* ptr = data; ptr != end; ptr++) {
    (*ptr) = value;
  }
}
```

```
/* Continuation (Page #4) of Image.tpp */

template<typename PIXEL>
template<typename OTHER_PIXEL>
void Image<PIXEL>::Copy(const Image<OTHER_PIXEL>& source_image)
{
  OTHER_PIXEL* src_ptr = source_image.data;
  PIXEL* end = data + size;
  for(PIXEL* dest_ptr = data;
      dest_ptr != end;
      dest_ptr++, src_ptr++) {
    (*dest_ptr) = static_cast<PIXEL>(*src_ptr);
  }
}

template<typename PIXEL>
Window<PIXEL> Image<PIXEL>::GetWindow(unsigned int start_row,
                                      unsigned int start_col,
                                      unsigned int height,
                                      unsigned int width)
{
  return Window<PIXEL>(*this, data, start_row,
                       start_col, height, width);
}
#endif // IMAGE_TPP_DEFINED
```

```
/* File Window.tpp */
#ifndef WINDOW_TPP_DEFINED
#define WINDOW_TPP_DEFINED
#include<Image.tpp>

template<typename PIXEL>
class Window : public Image<PIXEL> {
public:
  typedef ImageIterator<PIXEL> iterator;

  Window(Image<PIXEL>& src, PIXEL* data_ptr,
         unsigned int start_row, unsigned int start_col,
         unsigned int height, unsigned int width);
  virtual ~Window<PIXEL>() { this->data = NULL; }

  Window<PIXEL>& operator = (const Window<PIXEL>& src);

  virtual unsigned int EndOfRowIncrement() const
  {return end_of_row_increment;}

protected:
  unsigned int end_of_row_increment;
};

template<typename PIXEL>
Window<PIXEL>::Window(Image<PIXEL>& src, PIXEL* data_ptr,
                      unsigned int start_row, unsigned int start_col,
                      unsigned int height, unsigned int width)
{
  this->rows = height;
  this->cols = width;
  this->size = height * width;
  end_of_row_increment = src.Cols() - width + 1;
  this->data = data_ptr + (src.Cols() * start_row) + start_col;
  this->row_ptrs = new PIXEL*[height];
  PIXEL* data_iter = this->data;
  PIXEL** end_iter = this->row_ptrs + height;
  for(PIXEL** row_iter = this->row_ptrs;
      row_iter != end_iter;
      row_iter++, data_iter += src.Cols()) {
    (*row_iter) = data_iter;
  }
}
```

```
/* Continuation of File Window.tpp */
template<typename PIXEL>
Window<PIXEL>& Window<PIXEL>::operator = (const Window<PIXEL>& src)
{
  this->rows = src.rows;
  this->cols = src.cols;
  this->size = src.size;
  end_of_row_increment = src.end_of_row_increment;
  this->data = src.data;
  this->row_ptrs = new PIXEL*[this->rows];
  PIXEL* data_iter = this->data;
  PIXEL** end_iter = this->row_ptrs + this->rows;
  for(PIXEL** row_iter = this->row_ptrs;
      row_iter != end_iter;
      row_iter++, data_iter += (this->cols + end_of_row_increment)) {
    (*this->row_ptrs) = data_iter;
  }
}

#endif //  WINDOW_TPP_DEFINED
```

```
/* File ImageIterator.tpp */
#ifndef IMAGE_ITERATOR_TPP_DEFINE
#define IMAGE_ITERATOR_TPP_DEFINE
#include<cstring>
#include<iostream>
using std::cout;
using std::endl;

template<typename PIXEL>
class ImageIterator {
public:
  ImageIterator(PIXEL* start, unsigned int columns, unsigned int
eof_incr)
  : ptr(start), cols(columns), end_of_row_increment(eof_incr), ctr(0) {}
  ~ImageIterator() {ptr = NULL;}

  PIXEL& operator * () {return *ptr;}
  ImageIterator<PIXEL>& operator ++ ();
  bool operator == (const ImageIterator<PIXEL>& other)
  {return (ptr == other.ptr);}
  bool operator != (const ImageIterator<PIXEL>& other)
  {return (ptr != other.ptr);}

protected:
  PIXEL* ptr;
  unsigned int cols;
  unsigned int end_of_row_increment;
  unsigned int ctr;
};

template<typename PIXEL>
ImageIterator<PIXEL>& ImageIterator<PIXEL>::operator ++ ()
{
  if ((ctr % cols) != 0) {
    ptr++;
    ctr++;
  }
  else {
    ptr += end_of_row_increment;
    ctr = 0;
  }
  return *this;
}
#endif // IMAGE_ITERATOR_TPP_DEFINE
```