CSCI-1200 Data Structures — Spring 2015
Test 1 — Solutions

1 Opening a New Hair Salon [/32]

In this problem you will implement a simple class named `Customer` to keep track of customers at a hair salon. First, we create 6 `Customer` objects:

```
Customer betty("Betty");     Customer chris("Chris");     Customer danielle("Danielle");
Customer erin("Erin");       Customer fran("Fran");       Customer grace("Grace");
```

Then, we can track customers as they come to the salon for appointments on specific dates with one of the salon's stylists. We use the `Date` class we discussed in Lecture 2. You may assume the appointments are entered chronologically, with increasing dates.

```
betty.hairCut   (Date(1,15,2015), "Stephanie");
chris.hairCut   (Date(1,17,2015), "Audrey"   );
grace.hairCut   (Date(1,20,2015), "Stephanie");
danielle.hairCut(Date(1,28,2015), "Stephanie");
chris.hairCut   (Date(2, 5,2015), "Audrey"   );
betty.hairCut   (Date(2, 9,2015), "Stephanie");
fran.hairCut    (Date(2,12,2015), "Audrey"   );
danielle.hairCut(Date(2,18,2015), "Lynsey"   );
betty.hairCut   (Date(2,20,2015), "Stephanie");
```

In the system, each customer record will store the customer's preferred stylist. The preferred stylist is defined as a customer's most recent stylist. A message is printed to `std::cout` on each customer's first visit to the salon, or if a customer switches to a new stylist. Here is the output from the above commands:

```
Setting Stephanie as Betty's    preferred stylist.
Setting Audrey    as Chris's    preferred stylist.
Setting Stephanie as Grace's    preferred stylist.
Setting Stephanie as Danielle's preferred stylist.
Setting Audrey    as Fran's     preferred stylist.
Setting Lynsey    as Danielle's preferred stylist.
```

Next, we insert the customers into an STL `vector`:

```
std::vector<Customer> customers;
customers.push_back(betty);  customers.push_back(chris);  customers.push_back(danielle);
customers.push_back(erin);   customers.push_back(fran);   customers.push_back(grace);
```

And then sort & print them first alphabetically by stylist, and secondarily by most recent visit to the salon:

```
std::sort(customers.begin(),customers.end(),stylist_then_last_appointment);
for (int i = 0; i < customers.size(); i++) {
   std::cout << customers[i].getName() << " has had "
             << customers[i].numAppointments() << " appointment(s) at the salon";
   if (customers[i].numAppointments() > 0) {
      std::cout << ", most recently with " << customers[i].getStylist()
                << " on " << customers[i].lastAppointment(); }
   std::cout << "." << std::endl;
}
```

Which results in this output to the screen:

```
Erin     has had 0 appointment(s) at the salon.
Chris    has had 2 appointment(s) at the salon, most recently with Audrey    on  2/ 5/2015.
Fran     has had 1 appointment(s) at the salon, most recently with Audrey    on  2/12/2015.
Danielle has had 2 appointment(s) at the salon, most recently with Lynsey    on  2/18/2015.
Grace    has had 1 appointment(s) at the salon, most recently with Stephanie on  1/20/2015.
Betty    has had 3 appointment(s) at the salon, most recently with Stephanie on  2/20/2015.
```

Note: Don't worry about output formatting/spacing. You may assume that the `Date` class has an `operator<` to compare/sort dates chronologically and an `operator<<` to print/output `Date` objects.

1.1 `Customer` Class Declaration [/15]

Using the sample code on the previous page as your guide, write the class declaration for the `Customer` object. That is, write the *header file* (`customer.h`) for this class. You don't need to worry about the `#include` lines or other pre-processor directives. Focus on getting the member variable types and member function prototypes correct. Use `const` and call by reference where appropriate. Make sure you label what parts of the class are `public` and `private`. Include prototypes for any related non-member functions. Save the implementation of all functions for the `customer.cpp` file, which is the next part.

Solution:

```
class Customer {
public:
  // CONSTRUCTOR
  Customer(const std::string& name);
  // ACCESSORS
  const std::string& getName() const;
  const std::string& getStylist() const;
  const Date& lastAppointment() const;
  int numAppointments() const;
  // MODIFIERS
  void hairCut(const Date &d,const std::string &stylist);
private:
  // REPRESENTATION
  std::string customer_name;
  std::string preferred_stylist;
  std::vector<Date> appointments;
};

// helper function for sorting
bool stylist_then_last_appointment(const Customer &c1, const Customer &c2);
```

1.2 `Customer` Class Implementation [/17]

Now implement the member functions and related non-member functions of the class, as they would appear in the corresponding `customer.cpp` file.

Solution:

```
// CONSTRUCTOR
Customer::Customer(const std::string &name) {
  customer_name = name;
}

// ACCESSORS
const std::string& Customer::getName() const {
  return customer_name;
}
const std::string& Customer::getStylist() const {
  return preferred_stylist;
}
const Date& Customer::lastAppointment() const {
  return appointments.back();
}
int Customer::numAppointments() const {
  return appointments.size();
}

// MODIFIER
void Customer::hairCut(const Date &d,const std::string &stylist) {
  if (stylist != preferred_stylist) {
    std::cout << "Setting " << stylist << " as " << customer_name << "'s preferred stylist." << std::endl;
    preferred_stylist = stylist;
  }
  appointments.push_back(d);
}
```

```
// COMPARISON FUNCTION FOR SORTING
bool stylist_then_last_appointment(const Customer &c1, const Customer &c2) {
  return (c1.getStylist() < c2.getStylist() ||
          (c1.getStylist() == c2.getStylist() && c1.lastAppointment() < c2.lastAppointment()));
}
```

2 Color Analysis of HW1 Images [/21]

Write a function named `color_analysis`, that takes three arguments: `image`, an STL `vector` of STL `string`s representing a rectangular ASCII image (similar to HW1); an integer `num_colors`; and a character `most_frequent_color`. The function scans through the image and returns (through the 2nd & 3rd arguments) the number of different colors (characters) in the image & the most frequently appearing color.

Solution:

```
void color_analysis(const std::vector<std::string> &image, int &num_colors, char &most_frequent_color) {
  // local variables to keep track of colors & counts
  std::vector<char> colors;
  std::vector<int> counts;
  // loop over every pixel in the image
  for (int i = 0; i < image.size(); i++) {
    for (int j = 0; j < image[i].size(); j++) {
      // add each pixel to the color counts
      bool found = false;
      for (int k = 0; k < colors.size() && !found; k++) {
        if (image[i][j] == colors[k]) {
          counts[k]++;
          found = true;
        }
      }
      // if we haven't seen this color before...
      if (!found) {
        colors.push_back(image[i][j]);
        counts.push_back(1);
      }
    }
  }
  // loop over all of the colors to find the most frequent
  int max_count = 0;
  for (int k = 0; k < colors.size(); k++) {
    if (max_count < counts[k]) {
      max_count = counts[k];
      most_frequent_color = colors[k];
    }
  }
  // also set the num_colors "return value"
  num_colors = colors.size();
}
```

What is the order notation of your solution in terms of w & h, the width & height of the image, and c, the number of different colors in the image?

Solution: $O(w * h * c)$. This function is a simple triply-nested loop. Thus, the order notation is a product of the controlling variables for each loop.

3 Dynamic Tetris Arrays [/26]

3.1 HW3 `Tetris` Implementation Order Notation [/6]

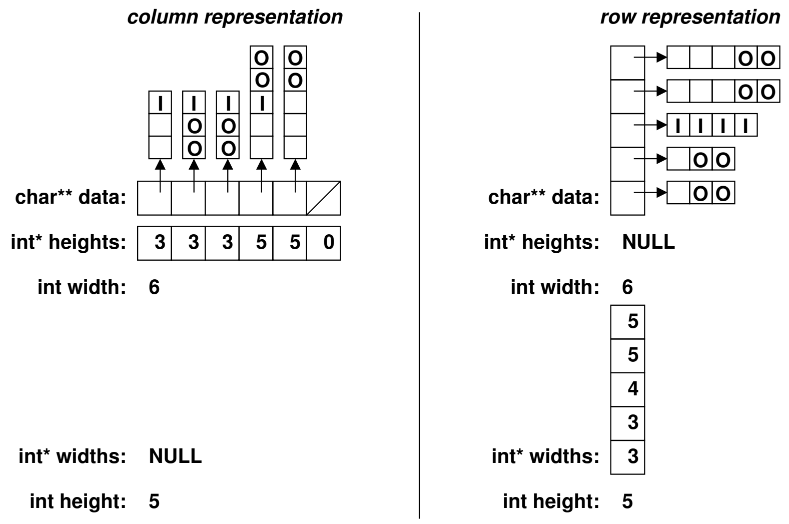
Grading Note: -1.5pts each unanswered or incorrect.

Match up the `Tetris` class member functions from HW3 with the appropriate order notation, where w is the width of the board and h is the maximum height of any column. Assume the solution is efficient, but uses only the 3 member variables specified in the original assignment (`data`, `heights`, and `width`).
Note: Some letters may be used more than once or not at all.

Solution:	**c**	`void add_piece(char piece,int rotation,int position);`	a)	$O(1)$
	a	`int get_width();`	b)	$O(w)$
	e	`int remove_full_rows();`	c)	$O(h)$
	b	`int get_max_height();`	d)	$O(w+h)$
	e	`void destroy();`	e)	$O(w*h)$

3.2 `Tetris` Representation Conversion [/20]

Now let's revisit the details of the dynamic memory representation for the game of Tetris. Your task is to convert a Tetris board from the *column representation* we used for HW3 to a *row representation.* In addition to the three member variables in our HW3 Tetris class: `data`, `heights`, and `width`, we add 2 additional member variables: `widths` and `height`. In the column representation we don't need the `widths` variable, so it is set to `NULL`. Each time the board is modified to add Tetris pieces or score full rows the `height` variable is updated as necessary to store the maximum height of any column.



The diagram above shows an example `Tetris` board first in *column representation* and then in *row representation* — the "before" and "after" diagrams for a call to the new `Tetris` class member function `convert_to_row_representation`. Note that once in row representation the `heights` variable isn't needed and we set it to `NULL`.
The `convert_to_row_representation` function takes no arguments.

Now write the `Tetris` class member function `convert_to_row_representation` as it would appear in the `tetris.cpp` implementation file. You may assume that before the call the board is in the column representation and the member variables are all set correctly. Make sure your code properly allocates new memory as needed and does not have memory leaks.

Solution:

```
void Tetris::convert_to_row_representation() {
  // allocate the top level arrays
  widths = new int[height];
  char** tmp = new char*[height];
  // for each row...
  for (int h = 0; h < height; h++ ) {
    // calculate the width of each row
    widths[h] = 0;
    for (int w = 0; w < width; w++ ) {
      if (heights[w] > h && data[w][h] != ' ') widths[h] = w+1;
    }
    // allocate a row of the correct width in the tmp structure
    assert (widths[h] > 0);
    tmp[h] = new char[widths[h]];
    // fill in the row character data
    for (int w = 0; w < widths[h]; w++) {
      if (heights[w] > h)
        tmp[h][w] = data[w][h];
      else
        tmp[h][w] = ' ';
    }
  }
  // cleanup the old structure
  delete [] heights;
  heights = NULL;
  for (int i = 0; i < width; i++) {
    delete [] data[i];
  }
  delete [] data;
  // point to the new data
  data = tmp;
}
```

4 Mystery Recursion [/9]

Grading Note: -2pts each unanswered or incorrect.

For each function or pair of functions below, choose the letter that best describes the program purpose or behavior.

A) infinite loop	E) function is not recursive	I) reverse the digits
B) factorial	F) sum of the digits	J) multiplication
C) integer power	G) syntax error	K) greatest common divisor
D) the answer is 42	H) modulo 2	L) other

Solution: K

```
int mysteryONE(int x, int y) {
  if(y == 0)
    return x;
  else
    return mysteryONE(y, x % y);
}
```

Solution: F

```
int mysteryTWO(int x) {
  if (x == 0)
    return 0;
  else
    return mysteryTWO(x/10)
           + x%10;
}
```

Solution: H

```
int mysteryTHREEa(int x);

int mysteryTHREEb(int x) {
  if (x == 0)
    return 1;
  else
    return mysteryTHREEa(x-1);
}

int mysteryTHREEa(int x) {
  if (x == 0)
    return 0;
  else
    return mysteryTHREEb(x-1);
}
```

Solution: J

```
int mysteryFOUR(int x, int y) {
  if (x == 0)
    return 0;
  else
    return y +
      mysteryFOUR(x-1,y);
}
```

Solution: I

```
int mysteryFIVEa(int x, int y) {
  if (x == 0)
    return y;
  else
    return mysteryFIVEa
       (x/10, y*10 + x%10);
}

int mysteryFIVEb(int x) {
  return mysteryFIVEa(x,0);
}
```

Solution: B

```
int mysterySIX(int x) {
  if (x == 0)
    return 1;
  else
    return x *
        mysterySIX(x-1);
}
```

5 iClicker Replay [/9]

Grading Note: -2pts each unanswered or incorrect.

5.1 Which of the following statements is *true*?

Solution: D

(A) Students should memorize the specification for the STL `string` and `vector` data types.

(B) In order to get credit for Data Structures homework, you may not use an IDE – instead you must only compile your code using g++ 4.8 on the command line.

(C) If the compiler gives you warnings it will not produce an executable to run.

(D) When you make changes to your C++ code, you must re-compile the program in order to see changes when you run the executable.

(E) If your code runs perfectly (always gets the right answer on the homework submission server), it is *unnecessary* to use comments and good variable names, use logical program structure, or follow requirements listed in the homework .pdf about use of certain data structures or algorithms.

5.2 What is *not* a reason for making the member variables of a C++ class private?

Solution: B

(A) Because the instructor said we should always* make them private.

(B) If you don't specify, everything about a C++ `class` will be public. In contrast, by default, everything is private for a `struct` (inherited from the C programming language).

(C) It is a security measure that prevents accidental or erroneous modification of the object by external users of the class.

(D) It facilitates future changes to the internal representation to improve efficiency or add new functionality.

(E) This is an *abstraction barrier*, meaning the external user does not need to understand the details of the internal representation.

5.3 Which statement is true about good software engineering class design?

Solution: E

(A) Advanced programmers will always agree on the same ideal class design for a specific problem.

(B) You should always define both "accessor" (read) and "set" (write) member functions for each of the member variables of your class.

(C) You may only define one constructor for the new type defined by the class.

(D) Every custom class must define `operator<`, even if you don't ever need to sort objects of this type.

(E) None of the above.

5.4 Which of the following is a correct use of the *hash* or *pound* symbol, `#`, in C++?

Solution: A

(A) Simple if/else logic can be performed with the *preprocessor* before compilation & linking begins.

(B) When modules from STL or other libraries are used we include the library header file within "double quotes".

(C) When a user-defined type (C++ class) is used we include the its header file within <angle brackets>.

(D) It is the most convenient way to convert strings to numbers (integer or floating point).

(E) Just like Twitter, it is used to tag topics or modules within the program.

5.5 Which of the following statements is *false* about memory debuggers?

Solution: D

(A) Dr. Memory and/or Valgrind are available on Linux, MacOSX, and Windows platforms.

(B) Code that runs perfectly on a student's computer may still have a memory error.

(C) Step-by-step debuggers (like `gdb`, `lldb`, and your IDE debugger) and memory debuggers (like Dr. Memory and Valgrind) are complementary. They both play a significant role in the debugging process.

(D) A memory debugger points to the line number in the code that must be edited to fix the memory leak.

(E) Even though modern computers have an obscene amount of RAM and harddrive space, it is still important to use a memory debugger to ensure all dynamically-allocated memory has been deleted.

5.6 Which of the following is *not* a use of the `&` symbol in C++?

Solution: B

(A) To indicate `pass-by-reference` in the argument list of a function.

(B) It means "follow that pointer" when placed in front of an object of pointer type.

(C) It prevents unnecessary copying of large data structures when passing data to or returning data from functions. (But it must be used appropriately.)

(D) It means "take the address of" when placed in front of a variable or l-value.

(E) `&` is the *bitwise and* operator. (It compares two input numbers bit-by-bit and sets each bit of the output to '1' only if both input numbers have '1' at that same bit position. Otherwise that bit is set to '0'.) Note: The `&&` operator is *logical and*. (We use this in `if` statements to check if 2 (or more) things are true.)