CSCI-1200 Data Structures — Spring 2019
Test 1 — Thursday, January 31st 6-7:50pm

Xinhao Luo	luox6@rpi.edu lab section: 11
room: Sage 3303 zone: BETA row: 10 seat: 1	6-7:50pm

Write the name of one of your undergraduate mentors.

Eli

What time does your lab section end at?

1:50 P.m.

- This exam has 4 problems worth a total of 100 points (including the cover sheet).
- This packet contains 9 pages of problems numbered 1-9. Please count the pages of your exam and raise your hand if you are missing a page.
- The packet contains 1 blank pages. If you use a blank page to solve a problem, make a note in the original box and clearly label which problem you are solving on the blank page.
- This test is closed-book and closed-notes except for the .pdf notes you (optionally) uploaded to Submitty by last night. These notes are the last 2 pages of your exam packet.
- **DO NOT REMOVE THE STAPLE OR SEPARATE THE PAGES OF YOUR EXAM. DOING SO WILL RESULT IN A -10 POINT PENALTY!**
- You may have pencils, eraser, pen, tissues, water, and your RPI ID card on your desk. Place everything else on the floor under your chair. Electronic equipment, including computers, cell phones, calculators, music players, smart watches, cameras, etc. is not permitted and must be turned "off" (not just vibrate).
- Please read each question carefully. Raise your hand if you have a question.
- Please state clearly any assumptions that you made in interpreting a question. Unless otherwise stated you may use any technique that we have discussed in lecture, lab, or on the homework.
- Please write neatly. If we can't read your solution, we can't give you full credit for your work.
- You do not need to write `#include` statements for STL libraries. Writing `std::` is optional.

Xinhao Luo luox6@rpi.edu

1 Short Answer Round [/ 16]

For each of the following statements, write if it is true or false, and then write 1-2 *complete* sentences explaining why. Most of these statements are false.

1.1 *sizeof()* and Arrays [/4]

True or False Since *sizeof(x)* tells us how much memory the variable x takes, we can use *sizeof()* to find out how many elements are in an array.

False ~~It~~ array is a pointer to the first element, it cannot acquire array's size..

1.2 l-values [/4]

True or False 500 can be used as an *l-value.*

False
'500' is an r-value and cannot be represented as l-value since it is not started with an letter or symbol.

1.3 Number Types [/4]

True or False The following code will compile and print "true":

```
int x = 5;
float y = 7.2;
x = y;
if(x==7) std::cout << "true";
```

False
type float does not match with type int.
the code will not compile as expected.

1.4 Vector Usage [/4]

True or False The following code will compile and print "true" *Hint: The fill constructor arguments are correct, and you can assume the correct header files are included.*

```
std::vector<int> x(5,5);
std::vector<float> y(5,7.2);
x=y;
if(x[0]==7) std::cout << "true";
```

False.
x and y are vectors with different types. the code will not compile.

2 Image Flood Fill [/ 24]

A popular operation not written in in Homework 1 is "floodfill" ("paint bucket"). floodfill takes a starting position (x,y), where x is the row and y is the column, a vector of strings that is the image, and a fill character. The function will change all pixels with the same value as (x,y) that can be reached by making a path starting from (x,y) and using only adjacent pixels (no diagonals) with the same value. Input pixel values will not be whitespace. Fill in the blanks in the code below to complete the *floodfill()* function.

```
starting_image:      floodfill(0,6,'Z',starting_image):      floodfill(3,2,'Z',starting_image):
   .....XX.                .....ZZ.                                .....XX.
   .....XX.                .....ZZ.                                .....XX.
   .XXX...X                .XXX...X                                .ZZZ...X
   ..XXXX..                ..XXXX..                                ..ZZZZ..
   ..X..X..                ..X..X..                                ..Z..Z..
```

```
[ void ] floodfill(int x, int y, [ char ] fill_char, [ vector<string> ] image) {
  char old_char = image[x][y];
  [ image[x][y] ] = ' '; //mark the initial pixel
  while(1) {
    int tmp = 0;
    for (unsigned int i = 0; i < [ image.size() ]; i++) {
      for (unsigned int j = 0; j < [ image[j].size() ]; j++) {
        if (image[i][j] == [ old_char ]) {
          // for any pixel matching the original character, see if it neighbors a temporarily marked pixel
          // legal_and_match(image,x,y,c) returns true if pixel (x,y) is within bounds and has value c
          if (legal_and_match(image, [ x+1, y, ' ' ]) ||
              legal_and_match(image, [ x, y+1, ' ' ]) ||
              legal_and_match(image, [ x-1, y, ' ' ]) ||
              legal_and_match(image, [ x, y-1, ' ' ])) {
            tmp++;
            [ image[i][j] ] = ' ';
          }
        }
      }
    }
    // if no pixels were changed, break out of the outer loop
    if (tmp == 0) break;
  }
  // replace all temporary pixels with the foreground char (assume this is implemented)
  replace(' ',fill_char,image);
}
```

3 Laundry Baskets [/ 43]

For this problem you will be writing two classes, `Basket` and `Clothing`. The `Basket` holds zero or more pieces of `Clothing` and has a number to identify it. A `Clothing` object has an ID to identify it, and is either dirty or clean. All `Clothing` starts dirty. Here is an example code segment that uses the two classes:

```
Basket b1(1), b2(24);
PrintBasket(b1);
Clothing c1("white socks");
b1.addToBasket(c1);
b2.addToBasket(c1);
PrintBasket(b2);
b1.washClothes();
b1.addToBasket(Clothing("ugly christmas sweater"));
PrintBasket(b1);
PrintBasket(b2);
```

And here is the output:

```
Basket 1 has 0 clothes:
Added white socks to basket 1
Added white socks to basket 24
Basket 24 has 1 clothes:
  white socks (dirty)
Washing white socks
Added ugly christmas sweater to basket 1
Basket 1 has 2 clothes:
  white socks (clean)
  ugly christmas sweater (dirty)
Basket 24 has 1 clothes:
  white socks (dirty)
```

3.1 `Clothing` Declaration (*Clothing.h*) [/ 11]

Start by writing the header file for the `Clothing` class.

```
class Clothing {
public:
    Clothing(const string &id);
    const string & getID() const;
    bool getState() const;
    void setState(bool state);

private:
    string ID_;
    bool state_;
```

sample solution: 13 line(s) of code

3.2 `Clothing` Implementation (*clothing.cpp*) [/ 6]

Next write the implementation of the `Clothing` class.

```
Clothing::Clothing(const string &id) {
    ID_ = id;
    state_ = fake;
}

const string &Clothing::getID() const {
    return ID_;
}

bool Clothing::geeState() const {
    return state_;
}

Void Clothing::seeState(bool state) {
    state_ = state;
}
```

sample solution: 25 line(s) of code

3.3 `Basket` Declaration (*Basket.h*) [/ 13]

Next write the header file for the `Basket` class.

```
#include "clothing.h"
class Basket {
public:
    Basket(int ID);
    int getID() const;
    vector<Clothing> getClothes() const;
    void addToBasket(Clothing &c);
    void washClothes();

private:
    int ID_;
    vector<Clothing> clothes;

};
void PrintBasket(const Basket &b);
```

sample solution: 17 line(s) of code

3.4 `Basket` Member Functions (*basket.cpp*) [/ 8]

Write the implementation of the member functions of the `Basket` class.

```
#include "Clothing.h"
#include "Basket.h"
Basket::Basket(int ID){
    ID_ = ID);
}

int Basket::getID() const {
    return ID_;
}

vector<Clothing> Basket::getClothes() const {
    return clothes;
}

void Basket::addToBasket(Clothing& c) {
    clothes.push_back(c);
    cout << "Added " << c.getID()
         << " to basket " << ID_ << endl;
}
void Basket::washClothes() {
    for (unsigned int i=0; i<clothes.size(); i++) {
        clothes[i].setState(true);
        cout << "Washing " << clothes[i].getID()
             << endl;
    }
}
```

sample solution: 20 line(s) of code

3.5 `Basket` Non-Member Functions (*basket.cpp*) [/ 5]

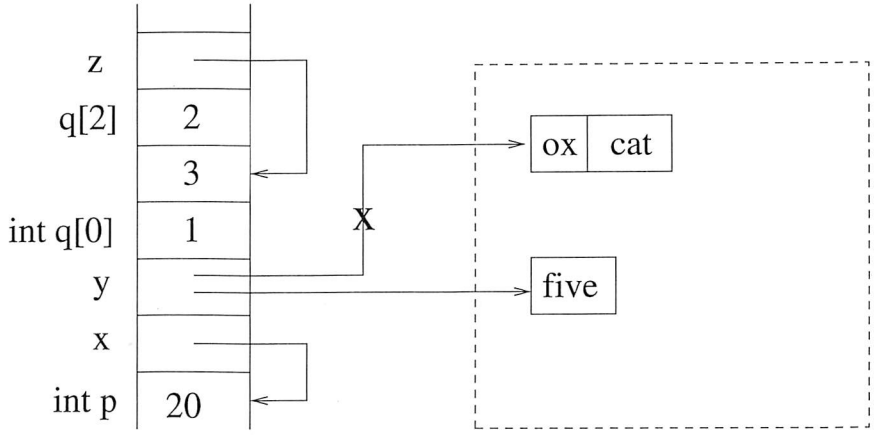
Finally, implement any non-member functions that were declared in *Basket.h* . You do not need to rewrite any `#include` statements written in Section 3.4 .

```
void PrintBasket(const Basket &b) {
    cout << "Basket " << b.getID()
         << " has " << b.getClothes().size()
         << " clothes:" << endl;

    vector<Clothing> all = b.getClothes();

    for (unsigned int i=0; i < all.size(); i++) {
        cout << "  " << all[i].getID()
             << " C";
        if (all[i].getSeated()) {
            cout << "lean";
        } else {
            cout << "irty";
        }
        cout << ")" << endl;
    }
}
```

sample solution: 15 line(s) of code

4 Memory Coding [/ 14]

Write code to produce the memory diagram shown on this page. An 'X over a line denotes a pointer that has been replaced by a more recent pointer. Some types have been left out.




```
int * z;
int q[3] = {1, 2, 3};
*z = &q[1];
string *y = new string[2]();
y[0] = "ox";
y[1] = "cat";
y = new string("five");
int *x;
int p = 20;
*x = &p;
```

sample solution: 12 line(s) of code

After code is run to produce the memory diagram shown above, is it possible to clean up all dynamically allocated memory? If it is not, write 1-2 sentences explaining why. If it is possible, write the code that will clean up the heap.

It is not possible to clean up all since we lose the pointer of string array "ox cat", which cannot be clean afterwards.

(blank page)

Xinhao Luo luox6@rpi.edu

I - Basic

1.1 - #include

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cassert>
#include <algorithm>
#include <bits/stdc++.h> //include everything ↑
#include "NAME.h"
```

1.2 - Functions

1.2a - Forward

```
void printValues(int x, int y);
int main()
{ printValues(6, 7);
  return 0; }
void printValues(int x, int y)
{ std::cout << x << std::endl;
  std::cout << y << std::endl; }
```

1.3 - Header Files (.h)

```
#ifndef NAME_H
#define NAME_H
int yourFunction(int x, int y);

#endif
```

1.3a - Macro Defines

```
#define NUMBER 200
std::cout << "The number is: " <<NUMBER << std::endl;
```

II – Data Types

2.1 - Define

```
bool bValue;  char chValue;  int nValue;  float fValue;  double dValue;
std::vector<DATA_TYPE> YOUR_NAME;
```

2.2 - Initialization

```
int nValue = 5; // copy initialization
int nValue(5); // direct initialization
```

2.3 - Const

```
const double gravity { 9.8 };
gravity = 9.9; // not allowed, compile error
```

III – Operator

3.1 - Increment/decrement operators

```
++x   - Increment x, then evaluate x
—x    - Decrement x, then evaluate x
x++   - Evaluate x, then increment x
x—    - Evaluate x, then decrement x
```

3.3 - Logical operators

```
!x          - true if x is false, or false if x is true
x && y      - true if x and y are true, false otherwise
x || y      - true if x or y are true, false otherwise
```

3.3 - Comma, and conditional operators

```
Comma Operator:       x, y      - Evaluate x then y, returns value of y
Conditional Operator: c ? x : y - If c is true then x, otherwise y
```

IV - Control Flow

4.1 - For Loops

```
for (unsigned int i=0; i < 10; i++)
{ std::cout << i << " "; }
```

4.2 - While Loops

```
int count = 0; while (count < 10)
{ std::cout << count << " ";
  ++count; }
```

V - Arrays, Strings, and Pointers

5.1 - Arrays

```
int prime[5]; // allocate 5 integer variables
prime[0] = 2; // The first element has index 0
OR
int prime[5] = { 2, 3, 5, 7, 11 }
```

5.2 - std::sort

```
const int length = 5;
int array[length] = { 30, 50, 20, 10, 40 };
std::sort(array, array+length);
for (int i=0; i < length; ++i)
   std::cout << array[i] << ' ';
```

5.3 - Selection Sort

```
const int length = 5;
int array[length] = { 30, 50, 20, 10, 40 };
for (int startIndex = 0; startIndex < length - 1; ++startIndex)
{  int smallestIndex = startIndex;
   for (int currentIndex = startIndex + 1; currentIndex < length; ++currentIndex)
   {  if (array[currentIndex] < array[smallestIndex])
      smallestIndex = currentIndex; }
   std::swap(array[startIndex], array[smallestIndex]); }
for (int index = 0; index < length; ++index)
   std::cout << array[index] << ' ';
```

5.4 - Pointers

5.4a - The address-of operator (&) and dereference operator (*)

```
x     // the value of variable x
&x    // the memory address of variable x
*&x   // the value at the memory address of variable x
```

5.4b - Declaring a pointer

```
int *iPtr1, *iPtr2;
```

5.4c - Dereferencing pointers

```
int value = 5;
&value          // address of value                                  - 0012FF7C
value           // contents of value                                 - 5
int *ptr = &value; // ptr points to value
ptr             // address held in ptr, which is &value              - 0012FF7C
*ptr            // dereference ptr (get the value that ptr is pointing to)  - 5
```

5.5 – Dynamic Memory

5.5a - Dynamically allocating single variables

```
new int;              // dynamically allocate an integer (and discard the result)
int *ptr = new int;   // dynamically allocate an integer and assign the address to ptr
*ptr = 7;             // assign value of 7 to allocated memory
```

VI - Input and output (I/O)

6.1 – Commons

Open File by using constructor

```
ifstream (const char* filename, std::ofstream::out | std::ofstream::app);
ifstream fin(filename, std::ofstream::out | std::ofstream::app)
ifstream fin("filename");
```

Open File by using open method

```
//Calling of default constructor
ifstream fin;
fin.open(filename, std::ofstream::out | std::ofstream::app)
fin.open("filename");
```

6.2 – Modes

Constant	*Stands*	*Access*
in *	input	File open for reading
out	output	File open for writing
ate	at end	The output position starts at the end of the file.
app	append	All output appending to its existing contents.
trunc	truncate	Discard any contents that existed in the file before it is open.

6.3 - Default Open Modes

ifstream	ios::in	Stream class to read from files
ofstream	ios::out	Stream class to write on files
fstream	ios::in | ios::out	Stream class to both read and write from/to files

VII – Class, Header file and Cpp file

7.1 – Name.cpp

```
#include <bits/stdc++.h>
#include "name.h"
using namespace std;

myClass::myClass(const std::string& v1, int v2, bool v3, float v4, char v5)
{ m_v1 = v1; m_v2 = v2; m_v3 = v3; m_v4 = v4; m_v5 = v5; }

void myClass::Function(int integer, float floatingPoint, string strings)
{ ##Content## }

bool sortByInt(const myClass& r1, const myClass& r2)
   { return r1.getV2() < r2.getV2(); }
```

7.2 – Name.h

```
#ifndef NAME_H
#define NAME_H

class myClass
{
 public:
   myClass(const std::string& v1, int v2, bool v3, float v4, char v5);
   const std::string& getV1() const { return m_v1; }
   int                getV2() const { return m_v2; }
   bool               getV3() const { return m_v3; }
   float              getV4() const { return m_v4; }
   char               getV5() const { return m_v5; }
   void Function(int integer, float floatingPoint, string strings);
 private:
   const std::string& m_v1;
   int                m_v2;
   bool               m_v3;
   float              m_v4;
   char               m_v5;
};
bool sortByInt(const myClass& r1, const myClass& r2);
#endif
```

VIII – Misc

8.1 – Command Line Arguments

```
g++ -Wall -g *.cpp -o a.out
int main(int argc, char* argv[])
```

8.2 – getline()

```
string in_file(argv[1]);
ifstream input(in_file);
while (input.good())
  { getline (input,line); cout << line << endl; }
```

8.3 – Sort & Erase (Vector)

```
sort( vector.begin(),vector.end() );
vector.erase( unique(vector.begin(),vector.end() ),
vector.end() );
```

8.4 – String Methods

```
string line = "This is a line.";
line.substr(10,4); //Print "line"
line.replace(10,4,"shit"); //"This is a shit."
```

```
line.size(); //15
line.find('s'); //3
```

8.5 – Other Methods

```
std::max(x, y)
std::min(x, y)
std::ceil(x)
std::floor(x)
```

```
std::abs(x)
std::sqrt(x)
std::swap(x, y)
std::reverse(v.begin(),v.end())
```

Xinhao Luo luox6@rpi.edu

TA: YingYi
Mentors:

Anshui
Eli
Trevor
Geogre

Patr