CS140 Midterm Exam. February 16, 2012. James S. Plank

In all of the example pieces of code, you may assume that all of the proper "**#include**" and "**using**" statements are at the top of the program – I'm skipping them to save on space. If I ask you to give output, show me everything, including spaces and newlines. If a program is in the file **xxx.cpp**, then you can assume that it has been compiled into the output program **xxx**. When you write a program, you may skip the "**#include**" and "**using**" statements.

Put your answers on the given answer sheets. You can work on other sheets of paper, but I do not want to see your work.

Question 1

Take a look at the following code snippet in **q1.cpp**:

q1.cpp

```
main()
{
  cout << "Luigi";
  cerr << "Binky";
  cout << "Thor";
  cout << endl;
  cerr << endl;
}
```

This gets compiled into the program **q1**.

Suppose that we type the following on the command line:

```
UNIX> q1 > out1.txt
```

Part A: What gets printed to the screen?
Part B: What is in the file **out1.txt**?

Now, suppose we execute it as follows:

```
UNIX> q1 out1.txt
```

Part C: What gets printed to the screen?
Part D: What is in the file **out1.txt**?

Question 2

Take a look at the following programs:

q2a.cpp

```
main()
{
  int i, j;

  i = 5; j = 6;

  cin >> i;
  cin >> j;
  printf("%d %d\n", i, j);
}
```

q2b.cpp

```
main()
{
  double i, j;

  i = 5.89; j = 6;

  cin >> i;
  cin >> j;
  printf("%.1lf %.1lf\n", i, j);
}
```

q2c.cpp

```
main()
{
  string i, j, k;

  i = "Fred";  j = "Luther";

  cin >> i;
  cin >> j;
  k = j;
  k[0]++;
  printf("%s %s %s\n", i.c_str(),
       j.c_str(), k.c_str());
}
```

And take a look at the following files.

a.txt

```
7
8.7
```

b.txt

```
10 11
Lola
```

c.txt

```
1.2
4
```

d.txt

```
Tyler
3
```

Part AA: What is the output of "**q2a < a.txt**"?
Part AB: What is the output of "**q2a < b.txt**"?
Part AC: What is the output of "**q2a < c.txt**"?
Part AD: What is the output of "**q2a < d.txt**"?

Part BA: What is the output of "**q2b < a.txt**"?
Part BB: What is the output of "**q2b < b.txt**"?
Part BC: What is the output of "**q2b < c.txt**"?
Part BD: What is the output of "**q2b < d.txt**"?

Part CA: What is the output of "**q2c < a.txt**"?
Part CB: What is the output of "**q2c < b.txt**"?
Part CC: What is the output of "**q2c < c.txt**"?
Part CD: What is the output of "**q2c < d.txt**"?

Question 3

Write a procedure called **stoivec()**, which takes a string as a parameter and returns a vector containing each word of the string that is an integer. The string may be extremely large, so take that into account.

Question 4

Write a procedure called s**vectos()**, which takes a vector of strings as a parameter and returns a string containing each string of the vector, in the order in which it appears in the vector, separated by a space. It should also convert all uppercase letters to lowercase, using arithmetic on chars. Use a reference parameter to pass the vector – however, your code should make sure that the vector is not modified.

Question 5

Write a procedure called **avg()**, which has one parameter, a vector called **v**. Each element of **v** is a vector of integers. Your job is the following – for each element **v[i]** of **v**, print the average of all of **v[i]**'s values. Print to three decimal places. If **v[i]** is empty, print **"Bad"**. Again, **v** may be large.

Question 6

A & B: What are the values 37 and 5*256+15 in hexadecimal?
C & D: What are the values 0x1a and 0x100 in decimal?

Question 7

Take a look at the following class declaration. This is in the file **airplane.h**. As demonstrated in lecture, we are going to implement the class in **airplane.cpp**, and we are going to program a **main()** that uses instances of the **Airplane** class in **airplane_main.cpp**.

```
class Airplane {
  public:
    Airplane(int number);
    void Fly();
    void Height(int &time);
    string pilot;
  protected:
    int altitude;
    int N;
};
```

Below are 18 statements. Tell me whether each is true or false. I don't want justif cation, just a T or F:

Statement A: The program in **airplane_main.cpp** can create an instance of an `Airplane` with `"Airplane a;"`
Statement B: The code in **airplane.cpp** has no guarantees about what `pilot` will be from one call to the next.
Statement C: The program in **airplane_main.cpp** can freely access and modify `N`.
Statement D: The program in **airplane_main.cpp** must call `Fly()` before it calls `Height()`.
Statement E: The program in **airplane.cpp** can access `N`, but it cannot modify it.
Statement F: The program in **airplane.cpp** cannot access `N` directly.
Statement G: The program in **airplane.cpp** can freely access and modify `N`.
Statement H: Since `Fly()` does not return anything, and since it has no parameters, it really cannot do anything.
Statement I: There is no good reason to have `time` be a reference variable.
Statement J: The code in **airplane.cpp** can assume that if it sets `pilot` to be `"Fred"` in the constructor, that it will still be `"Fred"` when `Height()` is called.
Statement K: The program in **airplane_main.cpp** cannot access `N` directly.
Statement L: The program in **airplane_main.cpp** can access `N`, but it cannot modify it.
Statement M: The program in **airplane_main.cpp** can create an instance of an `Airplane` with `"Airplane a(50);"`
Statement N: The program in **airplane_main.cpp** can create an instance of an `Airplane` with:
`"Airplane *a; a = new Airplane(50);"`
Statement O: There is a good reason to have `time` be a reference variable.
Statement P: The program in **airplane_main.cpp** can create an instance of an `Airplane` with:
`"Airplane *a; a = new Airplane(100.0);"`
Statement Q: If the program needs to read command line arguments, it will do so in **airplane_main.cpp** using `argc` and `argv`.
Statement R: If the program needs to read command line arguments, it will do so in **airplane_main.cpp** using either `cin` or `fin`.

Question 8

The following is a description of a Topcoder problem. Solve it.

Given a set of parts and a set of boxes, you are asked to determine how many of the parts can be packed into boxes. You can put at most one part into each box. In order for a part to f t into a box, the box must be the same size as or larger than the part. You follow the following process when packing the parts:

Choose the smallest part that has not been packed yet. Find the smallest empty box the part f ts into. If there is no such box, you cannot pack any more parts, and you stop. Pack the part into the box. If all parts have been packed, stop. Otherwise, continue with step 1.

You are given a `vector <int> partSizes` containing the sizes of the parts, and a `vector <int> boxSizes` containing the sizes of the boxes. Both vector <int>s will be sorted in non-descending order. Return an int representing the maximum number of parts you can pack by following the process above.

```
Class:PackingParts
Method signature:int PackingParts::pack(vector <int> partSizes, vector <int> boxSizes)

CONSTRAINTS
-partSizes will contain between 1 and 50 elements, inclusive.
-boxSizes will contain between 1 and 50 elements, inclusive.
-Both partSizes and boxSizes will be sorted in non-decreasing order.
-Each element in partSizes and boxSizes will be between 1 and 100, inclusive.
```

```
Example 0:

partsizes: {2,2,2}
boxSizes: {1,2,2,3}

Returns: 3
```

We have three parts of size 2. Two of them can be packed into boxes of size 2, and one into a box of size 3.

```
Example 1:

partSizes: {1,5}
boxSizes: {2,5}

Returns: 2
```

The part of size 1 goes into the box of size 2 and the part of size 5 goes into the box of size 5.

```
Example 2:

partSizes: {10,10,10,10}
boxSizes: {9,9,9,10,10,10}

Returns: 3
```

We have four parts of size 10, but only three boxes that can f t parts that big.

```
Example 3:

partSizes: {1,1,1,1}
boxSizes: {1,2,2,3,6,7}

Returns: 4
```

```
Example 5:

partSizes: {1,1,1,1}
boxSizes: {2,3,6}

Returns: 3
```

```
Example 5:

partSizes: {10,32,46,55,55,84,100}
boxSizes: {15,31,34,46,59,68,83,99}

Returns: 6
```