CSCI-1200 Data Structures — Spring 2018
Exam 1 — Solutions

1 Parcel Delivery [/ 35]

In the following problem you will finish the implementation of a program that is designed to keep track of several delivery drivers. Each driver is represented by a `Driver` object which has an ID, a name, a maximum capacity in kg that their vehicle can carry, and the packages they are currently carrying. Each package is represented by a `Parcel` object. For this problem, you can assume that all weights and capacities are integers, and that there will not be duplicate driver IDs.

First, here's `main.cpp` and the output that it produces:

```
#include "Driver.h"
#include "Parcel.h"
// print_drivers implemented, but not included in handout
// add_parcel written in 1.4

int main(){
   std::vector<Driver> drivers;
   drivers.push_back(Driver(124,"Chris",50));
   drivers.push_back(Driver(8,"Sam",150));
   drivers.push_back(Driver(35,"Taylor",200));

   print_drivers(drivers);
   add_parcel(drivers,Parcel("A7X",25),0);
   add_parcel(drivers,Parcel("A7X",25),8);
   add_parcel(drivers,Parcel("S41",126),8);
   add_parcel(drivers,Parcel("AK3",10),35);
   add_parcel(drivers,Parcel("P1",1),124);
   add_parcel(drivers,Parcel("P2",1),124);
   add_parcel(drivers,Parcel("P3",1),124);
   print_drivers(drivers);
   std::sort(drivers.begin(), drivers.end(), bySmallestWeight);
   print_drivers(drivers);
   return 0;
}
```

The output:

```
Driver Chris (#124) is carrying 0 of 50 kgs:
Driver Sam (#8) is carrying 0 of 150 kgs:
Driver Taylor (#35) is carrying 0 of 200 kgs:

Could not find driver #0
Added parcel A7X to driver #8
Failed to add parcel S41 to driver #8
Added parcel AK3 to driver #35
Added parcel P1 to driver #124
Added parcel P2 to driver #124
Added parcel P3 to driver #124
Driver Chris (#124) is carrying 3 of 50 kgs: #P1 (1) kg #P2 (1) kg #P3 (1) kg
Driver Sam (#8) is carrying 25 of 150 kgs: #A7X (25) kg
Driver Taylor (#35) is carrying 10 of 200 kgs: #AK3 (10) kg

Driver Chris (#124) is carrying 3 of 50 kgs: #P1 (1) kg #P2 (1) kg #P3 (1) kg
Driver Taylor (#35) is carrying 10 of 200 kgs: #AK3 (10) kg
Driver Sam (#8) is carrying 25 of 150 kgs: #A7X (25) kg
```

1.1 `Parcel` Class Declaration (`Parcel.h`) [/6]

Start by writing the class declaration in the `Parcel.h` file. The Parcel class should support the constructor used in `main.cpp`, and should have two accessors, *getWeight* and *getID*. For this problem, please do not use constructor initializer lists. You do not need to use include guards. Remember that one line functions can be written in the .h file.

Solution:

```
class Parcel{
public:
   Parcel(const std::string& id, int weight);
   int getWeight() const { return m_weight; }
   const std::string& getID() const { return m_id; }
private:
   int m_weight;
   std::string m_id;
};
```

1.2 `Parcel` Class Implementation (`Parcel.cpp`) [/5]

Now write the class implementation for the Parcel class in `Parcel.cpp`.

Solution:

```
#include "Parcel.h"

Parcel::Parcel(const std::string& id, int weight){
   m_weight = weight;
   m_id = id;
}
```

1.3 Completing the `Driver` Class Implementation (`Driver.cpp`) [/10]

Assume that the implementation of the Driver class is complete except for *getCurrentWeight* and *bySmallestWeight*. The .h file looks like this:

```
#include "Parcel.h"
class Driver{
public:
   Driver(int id, const std::string& name, int capacity);
   int getCapacity() const;
   const std::string& getName() const;
   int getID() const;
   const std::vector<Parcel>& getParcels() const;
   //getCurrentWeight definition would go here. Returns total weight the Driver is carrying.
   bool addParcel(const Parcel& p); //Returns true if parcel was added, false if it was too big.
private:
   int m_id;
   std::string m_name;
   int m_capacity;
   std::vector<Parcel> m_parcels;
};

//bySmallestWeight definition would go here. Used in main.cpp
```

Finish the .cpp file by implementing both of the missing functions:

Solution:

```
int Driver::getCurrentWeight() const{
   int ret = 0;
   for(unsigned int i=0; i<m_parcels.size(); i++){
      ret += m_parcels[i].getWeight();
   }
   return ret;
}
```

```
bool bySmallestWeight(const Driver& d1, const Driver& d2){
   return d1.getCurrentWeight() < d2.getCurrentWeight();
}
```

1.4 *add_parcel* Implementation (`main.cpp`) [/14]

Finally, write the *add_parcel* function that goes in `main.cpp`.

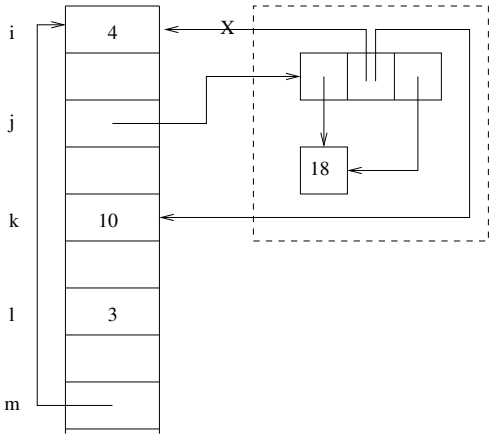
Solution:

```
void add_parcel(std::vector<Driver>& drivers, const Parcel& p, int driver_id){
   for(unsigned int i=0; i<drivers.size(); i++){
      if(drivers[i].getID() != driver_id) { continue; }
      if(drivers[i].addParcel(p)){
         std::cout << "Added parcel " << p.getID() << " to driver #"
                   << driver_id << std::endl;
         return;
      }
      else{
         std::cout << "Failed to add parcel " << p.getID() << " to driver #"
                   << driver_id << std::endl;
         return;
      }
   }
   std::cout << "Could not find driver #" << driver_id << std::endl;
}
```

2 Memory Diagramming [/ 26]

Consider the following code:

```
int i,**j,k,l,*m;
i = 0;
j = new int*[3];
j[0] = new int;
j[1] = &i;
m = *(j+1);
j[1] = &k;
k=10;
*(j[0]) = 5;
j[2] = j[0];
*(j[0]) = 18;
*m = 4;
l = 3;
```



2.1 Memory Diagram [/18]

First, draw a memory diagram for the above code. Do not erase lines for pointers that change, instead put an x over the middle of old line, and then draw the new pointer line.

2.2 Code Output [/8]

Next, write the output running this code will give:

```
std::cout << "i: " << i << std::endl;
std::cout << "j[0]: " << *j[0] << std::endl;
std::cout << "j[1]: " << *j[1] << std::endl;
std::cout << "j[2]: " << *j[2] << std::endl;
std::cout << "k: " << k << std::endl;
std::cout << "l: " << l << std::endl;
std::cout << "m: " << *m << std::endl;
```

Solution:

```
i: 4
j[0]: 18
j[1]: 10
j[2]: 18
k: 10
l: 3
m: 4
```

3 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.

3.1 Return Type [/4]

True or False If we are returning a string, we should always return using `const std::string&`.

Solution: False. If we are returning an automatic variable (non-member) then it will go out of scope so the reference would be invalid.

3.2 const Speed [/4]

True or False Using `const` types changes how fast the program runs.

Solution: False. This can produce compiler errors but has nothing to do with performance. const does not change if we copy a variable or not, and no run-time checking is done.

3.3 Reference Efficiency [/4]

True or False Passing by reference can be more efficient than passing by value.

Solution: True. Passing by reference means we just get a reference (like a pointer) to the original value. This means we don't have to make a second copy (saving time) or store a second copy (saving space).

3.4 const Members [/4]

True or False Every member function should have a `const` at the end of it. (e.g. `int get_var()` **const**).

Solution: False. Sometimes member functions need to alter the class's member variables.

4 Phrase Counting [/ 20]

In this problem you will write a function to count how many times a string appears inside a collection of other strings. Provided below is a code fragment which examines four strings: "banana", "bandana", "cabana", and "banabanbana". For this example, the output of the function is how many times each word had the letters "bana" consecutively in it. In this case, "bandana" has 0 instances of "bana" since the letter d gets in the way.

```
std::vector<std::string> words;
words.push_back("banana");
words.push_back("bandana");
words.push_back("cabana");
words.push_back("banabanabana");

std::vector<int> counts = count_phrase(words,"bana");
for(unsigned int i=0; i<counts.size(); i++){
  std::cout << words[i] << " contains \"bana\" " << counts[i]
            << " time(s)." << std::endl;
}
```

The expected output in this case:

```
banana contains "bana" 1 time(s).
bandana contains "bana" 0 time(s).
cabana contains "bana" 1 time(s).
banabanabana contains "bana" 3 time(s).
```

For this problem, the only STL string function you can use is *size()*. Do not use any C-style string functions (e.g. *strcmp()*).

Your answer should go in the box on the next page.

Write *count_phrase*:

Solution:

```
std::vector<int> count_phrase(const std::vector<std::string>& words, const std::string& phrase){
  std::vector<int> ret(words.size(),0);

  //Check each word
  for(unsigned int i=0; i<words.size(); i++){
    //Go letter by letter for starting position
    for(unsigned int j=0; j<words[i].size(); j++){
      unsigned int k;
      //Check if the substring is found starting at words[i][j+k]
      for(k=0; k<phrase.size() && j+k < words[i].size(); k++){
        if(words[i][j+k] != phrase[k]){
          break;
        }
      }

      //Found the whole phrase
      if(k==phrase.size()){
          ret[i]++;
      }
    }
  }

  return ret;
}
```