# CS140 Midterm Exam. April 5, 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

To the right is the queue header file, **queue.h**.

**Part A**: Draw me what the data structure looks like when you perform the following sequence of C++ statements. Use boxes, arrows and labels, as I did in lecture and in the lecture notes. I don't care what gets printed on standard output. Label what you can label.

```
Queue *q;

q = new Queue;
q.Push("Dub");
q.Push("Samson");
cout << q.Pop();
q.Push("Zak");
```

**Part B**: Implement the **Push()** method.

**Part C**: Implement the destructor. You will lose half of the points for this part if you use other methods in your implementation.

queue.h

```
#include <iostream>
using namespace std;

class Qnode {
  public:
    string s;
    Qnode *ptr;
};

class Queue {
  public:
    Queue();
    ~Queue();
    int Empty();
    int Size();
    void Push(string s);
    string Pop();
  protected:
    Qnode *first;
    Qnode *last;
    int size;
};
```

---

## Question 2

To the right is the output of the programs **acmhash** and **djbhash** on inputs "Jet" and "Naomi."

```
UNIX> echo Jet | acmhash
1264874752
UNIX> echo Naomi | acmhash
2439016044
UNIX> echo Jet | djbhash
193461032
UNIX> echo Naomi | djbhash
230264537
UNIX>
```

**Part A**: Suppose we are using open addressing with linear probing and **acmhash**, and a hash table whose size is 50. Tell me the first four indices one would look to find "Naomi," assuming that every lookup attempt ends up with a collision. In other words, give me four numbers.

**Part B**: Repeat part A, but using quadratic probing.

**Part C:** Repeat part A, but use double hashing, with **djbhash** as the second hash function.

**Part D**: Repeat part C, but look for "Jet" instead of "Naomi."

---

## Question 3

```
while (!x.empty()) x.erase(x.begin());
```

Suppose we execute the code snippet to the right. Below are six different data structures that **x** could be. Rank them from fastest to slowest in terms of running the code snippet. Just give me letters in your answer, like **A, B, C, D, E, F**.

**A.** A list with 500 elements
**B.** A vector with 1000 elements
**C.** A deque with 1500 elements
**D.** A deque with 2000 elements
**E.** A vector with 2500 elements
**F.** A list with 3000 elements

---

## Question 4

Write the program **setsort.cpp**, which prints the lines of standard input sorted on standard output, using either a set or a multiset. Your program should not strip duplicate lines.

## Question 5

Behold the program **q5.cpp** to the right. Tell me the output of the following commands:

**Command A:** `UNIX> echo 3 29 45 10 6 60 | q5`

**Command B:** `UNIX> echo 7 6 5 13 20 27 | q5`

**Command C:** `UNIX> echo 5 100 54 15 | q5`

**q5.cpp**

```
main()
{
  map <int, int> m;
  map <int, int>::iterator mit;
  int min, mod, i;

  cin >> mod;

  while (cin >> i) {
    i = i % mod;
    m[i]++;
  }

  for (mit = m.begin(); mit != m.end(); mit++) {
    i = mit->first;
    printf("%d: %d -> %d\n", i, m[i], m[(i+1)%mod]);
  }
}
```

---

## Question 6

To the right is a snippet of the header file from the Code Processor lab that you did.

Tell me two reasons why **Names** and **Phones** both have pointers to **Users** rather than **Users** in their second field.

```
class User {
  public:
     string username;
     string realname;
     int points;
     set <string> phone_numbers;
};

class Code_Processor {
  public:
    int New_Prize(string id, string description, int points, int quantity);
    int New_User(string username, string realname, int starting_points);
    int Delete_User(string username);

    int Add_Phone(string username, string phone);
    int Remove_Phone(string username, string phone);
    string Show_Phones(string username);

  protected:
    map <string, User *> Names;
    map <string, User *> Phones;
    set <string> Codes;
};
```

---

## Question 7 The following is a description of a Topcoder problem. Solve it.

You are going to stick the number of your room on the door. The shop near your house suggests wonderful sets of plastic digits. Each set contains exactly ten digits - one of each digit between 0 and 9, inclusive. Return the number of sets required to write your room number. Note that 6 can be used as 9 and vice versa.

```
DEFINITION
Class:RoomNumber
Method:numberOfSets
Parameters:int
Returns:int
Method signature:int numberOfSets(int roomNumber)

CONSTRAINTS
-roomNumber will be between 1 and
   1,000,000, inclusive.
```

```
EXAMPLE 0:

122

Returns: 2

Two sets are required
because each set
contains only one '2'
digit.
```

```
EXAMPLE 1:

9999

Returns: 2

Each set contains one '6'
digit and one '9' digit. '6'
could be used as '9' and
therefore two sets are enough.
```

```
EXAMPLE 2:

12635

Returns: 1
```

```
EXAMPLE 3

888888

Returns: 6
```