ECS 40 **Practice Final Key**

1. (30 points) This question will ask for a question about your p7 or p8 code. If you wrote your own code, then the answer should be easy and short.

2. (27 points) Write a recursive function, LargestFactor, which returns the largest factor of a number that is not the number itself. The syntax is : int LargestFactor(int number, int start_number). The recursion is always initiated with start_number set to 1. Here are some examples: LargestFactor(10, 1) is 5, LargestFactor(21, 1) is 7, LargestFactor(50, 1) is 25, LargestFactor(77, 1) is 11, and LargestFactor(7, 1) is 1. Since a number cannot be its own largest factor, the largest factor of a prime number is always 1. The heading is provided for you.

Pts	
0	`int LargestFactor(int number, int start_number) {`
4	`  if (start_number == number) // or start_number > number / 2`
4	`    return 1;`
6	`  int num = LargestFactor(number, start_number + 1);`
9	`  if( num == 1 && number % start_number == 0)`
2	`    return start_number;`
2	`  return num;`
	`}`

3. (10 points) Write a UNIX command line that will list the names of all the files in my current directory that use the STL. Only those files contain the phrase "using namespace std".

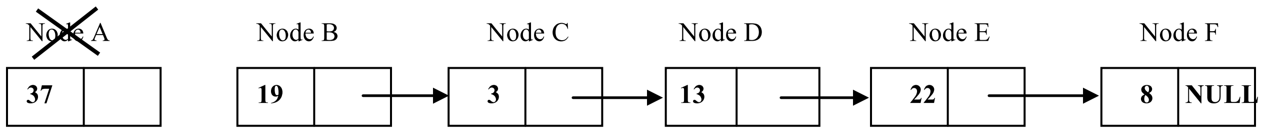
```
Pts:   3   1 1      2              1  2
       grep -l "using namespace std"  *
```

4. (30 points) Based on the following shell script, test.sh, answer the questions. Assume the following information

Line #	
1	`#! /bin/bash`
2	`i=0`
3	`for f in *.$1; do`
4	`  mv $f ${f%.*}.$2`
5	`  ((i = i + 1))`
6	`done`
7	`echo $i`

a) (5 points) What does line #1 do?
Informs the shell that bash should be used to process the script.
b) (5 points) What does line #3 do?
Selects every file in the current directory that ends with the first command line parameter.
c) (5 points) What does line #4 do?
Changes the name of a file ending with the first parameter so that its extension is now the second command line parameter.
d) (5 points) What does line #5 do?
It increments the shell variable i.
e) (10 points) What does the whole script do?
Searches for all files in the current directory that end with the first parameter, and replaces their extension with the second parameter. Lastly, it prints the number of files changed.

5. (20 points)
   a) (10 points) Assuming I have declared "queue<int, list<int> > Q". Do this by filling in the values and drawing the pointers (or entering NULL) for each list node that exists after the operations. Assume that the nodes are created in alphabetical order. If a node no longer exists, cross out its name, and make sure that it is not pointed to by an existing node. (Note: there may be more list nodes represented than needed.) Also indicate the letter of the list node that Front and Back would be pointing to.
   Q.push(37); Q.pop(), Q.push(19); Q.push(3); Q.push(13); Q.front(); Q.push(22); Q.push(8); Q.back();



FrontPtr: ____**B**___________ BackPtr:_____**F**_____ **-2 points for each incorrect Node, pointer, Front, or Back.**

   b) (10 points) Assuming I have declared "stack<int, vector<int> > st" and that the topIndex is initially -1. Show the contents of the stack after the following operations. (Note: there may be more positions than needed.) Also provide the final value of topIndex (the index).
   st.push(79); st.top(); st.push(73); st.push(32); st.push(94); st.top(), st.push(-2); st.pop(); st.push(7)

0	1	2	3	4	5	6	7	8
79	**73**	**32**	**94**	**7**				

topIndex: ____**4**____________ **-2 points for each incorrect value or Top.**

6. (190 points) For this section of the test you will be writing parts of a program that reads a passenger list for a train from a file, "train.txt", and then displays the information based on user requests. A sample file and its associated output are provided on a separate sheet. main(), the class PassengerCar, and the header for the Train class are also provided on a separate sheet.
   a) (76 points) The SleeperCar class is derived from PassengerCar and contains a map, passengerNames, that has pairs composed of keys passengerName<string> and elements berth <short>. The typedef of the map is SleeperMap. The SleeperCar class also has three methods: 1) a constructor, SleeperCar(int carNumber, int capac, int reserv); 2) void addName(string name, short berth = 0); and 3) void print(). Wirte a complete SleeperCar.h that provides all appropriate preprocessor directives and implementation code for the SleeperCar class so that there is no need for a SleeperCar.cpp file.

Pts	
2	`#ifndef SleeperCarH`
2	`  #define SleeperCarH`
2	`#include <map>`
2	`#include <iostream.h>`
2	`#include <iomanip.h>`
2	`#include "PassengerCar.h"`
3	`using namespace std;`
6	`typedef map <string, short> SleeperMap;`
4	`class SleeperCar : PassengerCar  // public, private, protected OK`
	`{`
2	`  SleeperMap passengerNames;`
1	`public:`
1	`  SleeperCar(int number, int cap, int res)`
6	`  :PassengerCar(2, number, cap, res){}`
1	`  void addName(string name, short berth = 0) {`
4	`    if(berth == 0)`
3	`      berth = ++reserved;`
7	`    passengerNames.insert(SleeperMap::value_type(name, berth));`
	`  }`
1	`  void print(){`
3	`    PassengerCar::print();`
11	`    for(SleeperMap::iterator itr = passengerNames.begin(); itr != passengerNames.end(); itr++)`
11	`      cout << " " << setiosflags(ios::left) << setw(20) << itr->first.c_str() << itr->second << endl; // c_str() may be omitted (but code won't work)`
	`  }`
1	`};`
1	`#endif`

b) (10 points) Provide the implementation code for the Train class print() method.

Pts	
1	`void Train::print() {`
6	`   for(int i = 0; i < count; i++)`
3	`      cars[i]->print();`
	`}`

c) (59 points) Provide the implementation code for the Train class >> operator. Note that you may not add any new methods to either PassengerCar, or SleeperCar.

Pts	
1	`istream& operator>> (istream &is, Train &train) {`
2	`  int type, number, capacity, reserved, berth;`
2	`  string name, firstName;`
6	`  while(is >> type >> number >> capacity >> reserved) {`
3	`    if(type == 1)`
9	`      train.cars[train.count++] = new PassengerCar(type, number, capacity, reserved);`
1	`    else {`
7	`      SleeperCar *sleeperPtr = new SleeperCar(number, capacity, reserved);`
5	`      for(int i = 0; i < reserved; i++) {`
4	`        is >> berth >> name >> firstName;`
3	`        name += " " + firstName;`
7	`        // or is >> berth; getline(is, name);`
4	`        sleeperPtr->addName(name, berth);`
	`      }`
4	`      train.cars[train.count++] = (PassengerCar*) sleeperPtr;`
	`    }`
	`  } // while more in file`
1	`  return is;`
	`}`

d) (20 points) Provide the implementation code for the Train class += operator. Note that the added passengers must be added to a Normal car that has room available to seat all of them.

Pts	
2	`void Train::operator+= (int number) {`
6	`  for(int i = 0; i < count; i++)`
8	`    if(cars[i]->getType() == 1 && cars[i]->getAvailable() >= number) {`
3	`      cars[i]->addPassengers(number);`
1	`      break;`
	`    } // if`
	`} // operator+=`

e) (25 points) If the array of PassengerCar pointers, cars, in the Train class was replaced with a TrainListNode *head, provide the implementation code for the Train class addSleeper()method. Note that the added passenger must be added to a Sleeper car that has a berth available for the passenger. You may assume that somewhere on the train there is a Sleeper berth available for the passenger. TrainListNode has the following definition.

```
class TrainListNode
{
   PassengerCar *info;
   TrainListNode *next;
public:
   TrainListNode(PassengerCar &in, TrainListNode *n);
   friend class Train;
};
```

Pts	
1	`void Train::addSleeper(string &name)  {`
8	`  for(TrainListNode *ptr = head; ptr; ptr = ptr->next)`
10	`    if(ptr->info->getType() == 2 && ptr->info->getAvailable() >= 1) {`
5	`       ((SleeperCar*)ptr->info)->addName(name);`
1	`       break;`
	`    }`
	`}`

6. (190 points) There are two formats to the lines in train.txt. For every car there is a line describing it: type <short>, 2) number, 3) capacity; and 4) number of seats already reserved. For type 2 cars, sleepers, the description line is followed by a list of passengers and their assigned berths: 1) berth <short>, and 2) name <string>. There is one of these lines for each reserved berth. You may assume that berths are filled sequentially, but the list may be in any order in the file.

```
% cat train.txt
1 8364 32 29
2 7920 12 5
4 Nixon, Richard
2 Carter, Jimmy
3 Carter, Rosalyn
1 Reagan, Ronald
5 Carter, Amy
1 4329 36 20

[davis@pc31 finalA]$ a.out
Your choice (0 = Done, 1 = Add Normal
Passengers, 2 = Add Sleepers): 1
Number of passengers: 1
Your choice (0 = Done, 1 = Add Normal
Passengers, 2 = Add Sleepers): 2
Number of passengers: 2
Name: Clinton, William
Name: Bush, George
Your choice (0 = Done, 1 = Add Normal
Passengers, 2 = Add Sleepers): 1
Number of passengers: 4
Your choice (0 = Done, 1 = Add Normal
Passengers, 2 = Add Sleepers): 0
8364 32 30
7920 12 7
 Bush, George        7
 Carter, Amy         5
 Carter, Jimmy       2
 Carter, Rosalyn     3
 Clinton, William    6
 Nixon, Richard      4
 Reagan, Ronald      1
0123456789012345678901234567890  // Added by
Sean for formatting purposes.
4329 36 24
[davis@pc31 finalA]$
```

```
int main()
{
  Train train;
  int choice, number;
  string name, dummy;
  ifstream inf("train.txt");
  inf >> train;

  do {
    cout << "Your choice (0 = Done, 1 = Add
Normal Passengers, 2 = Add Sleepers): ";
    cin >> choice;
    if(choice)
    {
      cout << "Number of passengers: ";
      cin >> number;
      getline(cin, dummy); // eats up '\n'
      if(choice == 1)
        train += number;
      else  // choice == 2
        for(int i = 0; i < number; i++)
        {
          cout << "Name: ";
          getline(cin, name);
          train.addSleeper(name);
        } // for each passenger
      } // if choice
  } while(choice);

  train.print();
  return 0;
}
```

In PassengerCar.h you would find:

```
class PassengerCar
{
protected:
  short type; // 1 = Normal, 2 = Sleeper
  int carNumber;
  int capacity;
  int reserved;

public:
  PassengerCar(short ty, int number, int cap, int
res):type(ty),
    carNumber(number), capacity(cap),
reserved(res){}
  virtual ~PassengerCar(){}
  virtual void print()
    {cout << carNumber << " " << capacity << " "
<< reserved << endl;}
  short getType()const {return type;}
  int getAvailable()const {return capacity -
reserved;}
  void addPassengers(int n){reserved += n;}
};
```

In Train.h you would find:

```
class Train
{
  PassengerCar *cars[10];
  int count;
public:
  Train():count(0){};
  void operator+= (int number);
  void addSleeper (string &name);
  friend istream& operator>> (istream
&is, Train &train);
  void print();
};
```