CS230 Data Structures Exam 1 – Devon J. Smith

1a.

Vectors – You would want to use a vector when you are not sure how many elements you will need. For example, to store user input that is used elsewhere in your program. The user would be able to input as many or as few objects into the list as they needed without getting an out of bounds exception due to a hard coded size (unless you specify the size of the vector in your code). The vector will automagically grow and shrink depending on how much data is stored in it.

List – You could use a list structure to store something such as a grocery list. The list is useful in that not only can you modify the data at the beginning or end of the list, but you can also insert and remove the data anywhere in between.

Stack – A stack can be used for last in first out situations, an example would be a deck of cards for a poker game. If your stack was used as the "draw" pile, you would shuffle your cards then load them into a stack. At that point, each card that was drawn would be taken from the top of the stack.

Queue – A cool example of using a deque in the real world would be to keep track of commands entered into a program. For example, the program needs to keep a record of the last 10 commands a user entered into a processing program, each time a user enters in a command, a copy of it is input into the front of the deque, when the 11th command is entered and pushed onto the front of the stack, the 1st command that was entered is popped from the back of the deque.

1b. The most beneficial objective of the SL to an application programmer is the fact that most of what they would need is already built into the STL. That means that a programmer can spend more time writing code and less time creating algorithms to do things like sort, manual memory management, etc. Not only that, but the SL gets vetted pretty heavily before new things are implemented which means that you can be pretty confident that the built in functions are as efficient as they can be. The biggest issue then is how efficient can the developer code their application to take advantage of these built in features?

1c. No I would not advocate buying an algorithm of this class unless there was no other possible way to make the algorithm more efficient. Here is why, if you have a sorter that was O(n^3) and you had to use it to sort an array with 100 elements, the array would have to pass through the loop 1,000,000 times before it was sorted. Compared to this, bubble sort, which is O(n^2), the array would only have to go through the loop 10,000 times, now that's a huge difference =.

1d. This means that, given the same type and amount of data (Where N == 1000 in this case), the O-Notation for Binary search would be O(3) and the sequential search O-Notation would be O(1000) meaning that if you were processing 1000 items, Binary search would be far more efficient than Sequential search. For Binary search, adding more items would cause the curve to rise more gradually as opposed to the Sequential search, which would curve more sharply in comparison.

2. The general array is similar to a vector in that it can store the same data types, but vectors differ in that they are implemented as a dynamic array and the memory they use is automatically managed.

3. True

4a. The node contains a field for data and one or more fields that contain a pointer to another node in the list (next, previous, etc.).

4b. Each node contains an address so that it can point to another node in the list, for example, in a linked list the pointer points to the next node in the list, or points to NULL if it's the last item.

4c. Accessing elements in an array is more efficient because you just need to specify the element you want to access and with lists, since you don't know where an object is stored, need iterators to walk through the list node by node to find an item.

On the flip side, if you want to add some data to the middle of a list, you just need to create a node and add a pointer to the next node, and change the pointer in the previous node to point to the new node. In an array, however, all of the data needs to be shifted over in order to make space for the new data you want to insert.

4d. In a circularly doubly linked list, the last node does not have a NULL pointer, instead, it points back to the first node of the list. An advantage to using this type of structure is that you can go to the end of the list by just traversing backwards, instead of traversing forward through the entire list to get to the end. A disadvantage is that if you don't code a way to remember where the first node is while you're traversing the list, an infinite loop could occur.

5a. vector<int>::iterator itr = someVector.begin();

5b. The increment operator causes the iterator pointer to be advanced to the next element, which is a requirement in order to traverse the elements in your container.

5c. If you are using a container of objects, you can use the following to access the objects attributes with an iterator: LItr->objectAttribute. If you have a container of say, integers, you can access the value of the element by dereferencing the iterator pointer like so: *LItr.

6. A restricted linear data structure is a data structure where you are restricted to what operations can be done to it. For example, when using a stack, you can only manipulate the top element.

7a. When a fellow programmer says that they are going to use a stack to implement an algorithm, this means that they are using a data structure for an algorithm that uses last in first out. This means that this algorithm will act like a person interacting with a pez dispenser, they are going to push some data into the structure then they are going to use top to read that element at the top of the stack, then pop that element off when they are done with it. That process will keep going until the stack is empty. All of the elements under the top element is hidden and no operations can be performed on them.

7b. The STL stack uses a deque as its underlying container if no other container class is specified.

7c. True

CS230 Data Structures Exam 1 – Devon J. Smith

8. See Attached.

9. A really important feature of a “safe stack” is the inclusion of exception handlers within the code in order to prevent the program from crashing if, say, a user tried to pop an object from an empty stack. Implementing a safe stack will allow the user to try and pop an item from an empty stack as many times as they possibly want, without crashing the program. A properly implemented exception will tell the user that the stack is empty, but will allow the program to continue running.

10. When someone mentions that they are using a queue to me, I think of a one way tunnel. With this data structure, you add data on one end and remove it on the other. This is also known as first in first out, and an example of its use would be in networking applications. Another example could be applied to a print server in a corporate environment. Each person sending a demotivational poster to be printed by the print server is called a “job”. So how do you organize all of these demotivational posters to be printed? Well you use a queue, each job you get goes into the queue and when the job in the front is finished, it’s popped and the print server reads the next job.

11a. The Queue is full

11b. #1: front == rear and fullFlag == true or false, then the queue is full or empty. #2: front == rear and itemsLoadedCount == arraySize, then the queue is full.

11c. True

11d. False

Bonus:

A. A skip list is an ordered list that has additional pointers within the nodes that point to other nodes in different parts of the list. The advantage of using a skip list is that it make for searching data faster and more efficient because you do not need to iterate through each node in order to search for something, instead, you can follow a pointer to a node further down the line. When doing this, if the target node contains a value that is greater than what you are searching for, then you backtrack and skip to another node until you find one that is closer to what you are searching for.

B. The first tactic to ensure failure in any programming class is to just not do any of the homework. The second tactic to fail a programming class is to make sure you write code, but make sure that it doesn’t work, or you could use a non standard library which in this class would be something like the Cpp Boost library. The reasons why we study data structures is to learn ways in which to handle large amounts of data and which structure we would use to best handle that data.

```
//***************************************************************************
// TITLE:                   Exam 1
// FILENAME:               main.cpp
// PREPARED FOR:           CS230
// PROGRAMMER(S):          Devon J. Smith
// DEVELOPMENT DATE:       08/03/14
// COMPILER USED:          Apple LLVM Version 5.1
// TARGET PLATFORM:        Mac OS X i386 & x86_64
//===================================================================
//                           PROJECT FILES
//    <LIST ALL PROGRAM AND HEADER FILES IN THE PROJECT HERE>
//        main.cpp (main)
//        Donations.cpp
//        Donations.h
//===================================================================
//        REVISION HISTORY
//   List revisions made to the Program
//
//   DATE     PROGRAMMER            DESCRIPTION OF CHANGES MADE
//    08/03/14 Devon J. Smith        Original
//===================================================================
//             PROGRAM DESCRIPTION
//  Gets a vector of people and calculates donations made. Also determines if the
    person is home
//  or not.
//
// INPUTS:   None
//
// OUTPUTS:  A list of people and how much they donated. Outputs total donations
    and expected donations
//
//===================================================================
//                          INCLUDE FILES
#include "Donations.h"
//
//===================================================================
//                        CONSTANT DEFINITIONS
//                             NONE
//
//===================================================================
//                      EXTERNAL CLASS VARIABLES
//                          NONE
//===================================================================

//***************************************************************************
//                       BEGINNING OF PROGRAM CODE
//***************************************************************************
using namespace std;

int main(int argc, const char * argv[])
{
    Donations d;
    vector<Donations> donationList;
    int expectedDonations = 0;
    int totalDonations = 0;
    srand((unsigned) time(NULL));
```

```cpp
    donationList.push_back(Donations("Nelsons", d.actualDonation(2)));
    donationList.push_back(Donations("Hills", d.actualDonation(1)));
    donationList.push_back(Donations("Kings", d.actualDonation(3)));
    donationList.push_back(Donations("Martins", d.actualDonation(1)));
    donationList.push_back(Donations("Moores", d.actualDonation(2)));
    donationList.push_back(Donations("Wilsons", d.actualDonation(1)));
    donationList.push_back(Donations("Davis", d.actualDonation(3)));
    donationList.push_back(Donations("Browns", d.actualDonation(2)));
    donationList.push_back(Donations("Williams", d.actualDonation(2)));
    donationList.push_back(Donations("Johnsons", d.actualDonation(2)));
    donationList.push_back(Donations("Smiths", d.actualDonation(1)));
    donationList.push_back(Donations("Jones", d.actualDonation(2)));

    for (int i = 0; i < donationList.size(); i++)
    {
        int home = rand() %10;
        if (home > 6)
        {
            expectedDonations += donationList[i].getDonation();
            cout << "The " << donationList[i].getName() << " are not home and
                    have been deffered." << endl;
        }
        else
        {
            expectedDonations += donationList[i].getDonation();
            totalDonations += donationList[i].getDonation();
            cout << "The " << donationList[i].getName() << " have donated $" <<
                    donationList[i].getDonation() << "." << endl;
        }
    }
    cout << "The expected total donation today was: $" << expectedDonations <<
          endl;
    cout << "The actual total donation today was: $" << totalDonations << endl;
    return 0;
}
/*
 The Nelsons are not home and have been deffered.
 The Hills have donated $3.
 The Kings have donated $68.
 The Martins have donated $4.
 The Moores have donated $5.
 The Wilsons have donated $4.
 The Davis have donated $98.
 The Browns are not home and have been deffered.
 The Williams have donated $5.
 The Johnsons have donated $5.
 The Smiths are not home and have been deffered.
 The Jones have donated $5.
 The expected total donation today was: $208
 The actual total donation today was: $197
 */
```

```
//==================================================================
//                           PROJECT FILES
//    <LIST ALL PROGRAM AND HEADER FILES IN THE PROJECT HERE>
//        main.cpp (main)
//        Donations.cpp
//        Donations.h
//==================================================================
//        REVISION HISTORY
//   List revisions made to the Program
//
//   DATE     PROGRAMMER            DESCRIPTION OF CHANGES MADE
//    08/03/14 Devon J. Smith        Original
//==================================================================
//                        CLASS DESCRIPTION
//  The donation object - the person....
//
//
//*********************************************************************************

//*********************************************************************************
//              PROCESS THIS FILE ONLY ONCE PER PROJECT
#ifndef __CS230_Exam1__Donations__
#define __CS230_Exam1__Donations__
//
//=======================================================================
//                       CONSTANT DEFINITIONS
//

//*********************************************************************************
//                  STANDARD AND USER DEFINED INCLUDES
#include <iostream>
#include <vector>
#include <time.h>
//*********************************************************************************
//                       USER DEFINED DATA TYPES
class Donations
{
public:
    Donations();
    Donations(std::string inName, int inDonation);
    std::string getName();
    int getDonation();
    int actualDonation(int povertyLevel);

private:
    std::string name;
    int donation;
};
//*********************************************************************************
//                      END OF CONDITIONAL BLOCK
#endif /* defined(__CS230_Exam1__Donations__) */
//*********************************************************************************
//                        END OF HEADER FILE
//*********************************************************************************
```

```
//===================================================================
//                           PROJECT FILES
//    <LIST ALL PROGRAM AND HEADER FILES IN THE PROJECT HERE>
//        main.cpp (main)
//        Donations.cpp
//        Donations.h
//===================================================================
//        REVISION HISTORY
//   List revisions made to the Program
//
//   DATE     PROGRAMMER            DESCRIPTION OF CHANGES MADE
//    08/03/14 Devon J. Smith        Original
//********************************************************************************
//                               CONSTANTS
//
//********************************************************************************
//                  STANDARD AND USER DEFINED INCLUDES
#include "Donations.h"
#include <vector>
#include <time.h>
using namespace std;
//********************************************************************************
//             Definition of member functions for class Entry
//********************************************************************************

string name;
int donation;

Donations::Donations() {}

Donations::Donations(string inName, int inDonation)
{
    name = inName;
    donation = inDonation;
}

string Donations::getName()
{
    return name;
}

int Donations::getDonation()
{
    return donation;
}

int Donations::actualDonation(int povertyLevel)
{
    int donationReturn = 0;

    if (povertyLevel == 1)
    {
        donationReturn = rand() %4 + 1;
    }
    else if (povertyLevel == 2)
    {
        donationReturn = 5;
    }
    else if (povertyLevel == 3)
    {
        donationReturn = rand() %95 + 5;
    }
    return donationReturn;
}
```