NWEN 241 Assignment #4 Sample Solution

There are many ways to solve the tasks in Assignment #4. This document provides one approach to solving the programming tasks.

Task 1

```
namespace dbms2
{
    class AbstractDbTable {
    public:
        virtual unsigned int rows() const = 0;
        virtual bool show(unsigned int r) const = 0;
        virtual bool add(struct movie &a) = 0;
        virtual bool remove(unsigned long id) = 0;
        virtual const struct movie *get(unsigned int r) const = 0;
        bool loadCSV(const char *infn);
        bool saveCSV(const char *outfn);
    };
}
```

Task 2

```
namespace dbms2
{
    class VectorDbTable : public AbstractDbTable {
    public:
        unsigned int rows() const;
        bool show(unsigned int r) const;
        bool add(movie &a);
        bool remove(unsigned long id);
        const movie *get(unsigned int r) const;

    private:
        // Vector of movies to hold records
        vector<movie> movies;
    };
}
```

Task 3

```
namespace dbms2
{
    struct node {
        struct movie data;
        struct node *next;
    };

    class LinkedListDbTable : public AbstractDbTable {
    public:
        unsigned int rows() const;
        bool show(unsigned int r) const;
        bool add(movie &a);
        bool remove(unsigned long id);
        const movie *get(unsigned int r) const;

        /**
         * We will need a constructor to initialize the private members
         */
        LinkedListDbTable();

        /**
         * We will need a deconstructor to free up dynamically allocated memory
         */
        ~LinkedListDbTable();

    private:
        /**
         * Linked list head
         */
        struct node *head, *tail;

        /**
         * Number of records
         */
        unsigned int count;
    };
}
```

Task 4

```
namespace dbms2
{
    unsigned int VectorDbTable::rows() const
    {
        return movies.size();
    }

    bool VectorDbTable::show(unsigned int r) const
    {
        if(r >= rows()) return false;

        movie m = movies[r];
        cout << "| " << m.id << " | " << m.title << " | " << m.year << " | " <<
m.director << " |" << endl;
        return true;
    }

    bool VectorDbTable::add(movie &m)
    {
        movies.push_back(m);
        return true;
    }

    bool VectorDbTable::remove(unsigned long id)
    {
        for(int i = 0; i < rows(); i++) {
            if(movies[i].id == id) {
                movies.erase(movies.begin()+i);
                return true;
            }
        }
        return false;
    }

    const movie *VectorDbTable::get(unsigned int r) const
    {
        if(r >= rows()) return NULL;

        return &movies[r];
    }
}
```

Task 5

```
    bool AbstractDbTable::loadCSV(const char *infn)
    {
        struct movie m;
        FILE *fp = fopen(infn, "r");

        if(fp == NULL) return false;

        while(!feof(fp)) {
            int r = fscanf(fp, "%lu,%[^,],%d,%[^\n]%*c", &m.id, m.title,
&m.year, m.director);
            if(r < 4) {
                fclose(fp);
                return false;
            }
            add(m);
        }

        fclose(fp);
        return true;
    }
```

Task 6

```
    bool AbstractDbTable::saveCSV(const char *outfn)
    {
        FILE *fp = fopen(outfn, "w");

        if(fp == NULL) return false;

        unsigned int i = 0;
        while(i < rows()) {
            const struct movie *m = get(i);
            if(m)
                fprintf(fp, "%d,%s,%d,%s\n", m->id, m->title, m->year, m-
>director);
            i++;
        }

        fclose(fp);
        return true;

    }
```

Task 7

```
namespace dbms2
{
    LinkedListDbTable::LinkedListDbTable()
    {
        head = NULL;
        tail = NULL;
        count = 0;
    }

    LinkedListDbTable::~LinkedListDbTable()
    {
        if(head) free(head);
        if(tail) free(tail);
    }

    unsigned int LinkedListDbTable :: rows() const
    {
        return count;
    }

    bool LinkedListDbTable :: add(movie &a)
    {
        struct node *new_node = (struct node*) malloc(sizeof(struct node));
        if (new_node ==  NULL) {
            cout <<"Memory allocation failed while creating new node\n";
            return false;
        }

        new_node->data = a;
        new_node->next = NULL;

        if (head == NULL) {
            // first node
            head = new_node;
            tail = new_node;
        } else {
            // non-first node
            tail->next = new_node;
            tail = new_node;
        }

        count++;

        return true;
    }

    bool LinkedListDbTable :: show(unsigned int r) const
    {
        if(r >= count) return false;

        struct node *tmp = head;
        movie m;
        int counter = 0;
        while (tmp != NULL) {
```

```
            if (counter == r) {
               m = tmp->data;
               break;
            }
            counter++;
            tmp = tmp->next;
        }

        cout << "| " << m.id << " | " << m.title << " | " << m.year << " | " <<
m.director << " |" << endl;

        return true;
    }

    bool LinkedListDbTable :: remove(unsigned long id)
    {
        struct node *tmp = head;
        movie m;
        bool flag = false;
        int counter = 1;

        while (tmp != NULL) {
            if ((tmp->data).id == id) {
                m = tmp->data;
                flag = true;
                break;
            }
            counter++;
            tmp = tmp->next;
        }

        if (!flag)
            return false;

        if (tmp->next == NULL && counter == count) {
            // tail node
            free(tail); // clearing the memory occupied by tail

            tmp = head;
            while(counter > 1) {
                tail = tmp;
                tmp = tmp->next;
                counter--;
            }
            tail->next = NULL;
        } else if (counter == 1) {
            // head node
            tmp = tmp->next; //moving to next node after former head
            free(head); // clearing the memory occupied by head

            head = tmp; //new head
            while (tmp != NULL) {
                tail = tmp;   //new tail
                tmp = tmp->next;
            }
```

```
        } else {
            // remaining nodes other than head or last node
            tail = head;

            while (counter > 2) {
               // moving the node pointer to one node before the node to be
deleted
               tail = tail->next;
                counter--;
            }

            tail->next = tail->next->next;  // linking the node before  the
deleted node to the node after the deleted node
        }

        count--;

        return true;
    }

    const movie *LinkedListDbTable::get(unsigned int r) const
    {
        if(r >= count) return NULL;

        struct node *tmp;
        tmp = head;
        int counter = 0;

        while (tmp != NULL) {
            if (counter == r)
               break;

            counter++;
            tmp = tmp->next;
        }

        return &tmp->data;
    }
}
```

Task 8

```
#include <iostream>
#include <regex>
#include "base_vdb.hh"

using namespace std;
using namespace base_dbms2;

int main(int argc, char *argv[])
{
    VectorDbTable *db;
    bool r;

    cout << "Instantiating VectorDbTable..." <<endl<<endl;
    db = new VectorDbTable();

    cout << "Invoking loadCSV(\"default.csv\")..." <<endl<<endl;
    r = db->loadCSV("default.csv");

    // Display all records if the first command line argument is showall
    if (std::string(argv[1]) == "showall")
    {
        cout << "Showing all records" <<endl;
        int i = 0;
        while(i < db->rows()) {
            db->show(i);
            i++;
        }
    }
    // Show record for the given row number
    if (std::string(argv[1]) == "show") {
        if (!argv[2])
            cout << "Second argument missing. Please enter the row number"
<<endl<<endl;
        else if (!regex_match(argv[2], regex("[+-]?[0-9]+")))
            cout << "Please input a number." << endl << endl;
        else {
            unsigned int r = atoi(argv[2]);

            if (r >= db->rows())
                cout << "Error: Row "<< r << " does not exist." <<endl<<endl;
            else
                db->show(r);
        }
    }

    cout << "Freeing VectorDbTable..." << endl;
    delete db;


    return 0;
}
```