Assignment 3

CST8333 19F

Professor: Stanley Pieda

By: Lucas Estienne (esti0011)

040 819 959

## Evidence of Learning

* 1. Variables:

// parse some values from strings

    cheeseId, err := strconv.ParseInt(line[0], 10, 64)

    if err != nil { cheeseId = 0 }

    fatContentPercent, err := strconv.ParseFloat(line[10], 32)

    if err != nil { fatContentPercent = 0.0 }

    moisturePercent, err := strconv.ParseFloat(line[11], 32)

    if err != nil { moisturePercent = 0.0 }

    organic, err := strconv.ParseBool(line[20])

    if err != nil { organic = false }

* 1. Loops:

// convert lines to records slice

    for i := 0; i < numRecords; i++ {

        records = append(records, lineToRecord(lines[i]))

    }

* 1. Decision Structure:

    if strings.TrimSpace(first) != "" {

        return first

    } else if strings.TrimSpace(second) != "" {

        return second

    } else {

        return "N/A"

    }

* 1. File I/O & functions:

// helper function to read CSV

func getLinesFromCSV(filePath string) (lines [][]string, err error) {

    // open file

    file, err := os.Open(filePath)

    check(err)

    defer file.Close() // defer closing the file until function returns

    // create CSV Reader from file

    reader := csv.NewReader(file)

    return reader.ReadAll()

}

* 1. Exception handling:

// helper function to do error handling

func check(e error) {

    if e != nil {

        log.Fatal("Error", e)

        panic(e)

    }

}

lines, err := getLinesFromCSV(filePath)

check(err)

\_, err := fmt.Scanf("%d", &selection)

        if err != nil {

            selection = 0

            fmt.Println("\nPlease enter a valid option.")

        } else if selection < 1 || selection > 8 {

            selection = 0

            fmt.Println("\nPlease enter a valid integer between 1 and 8.")

        }

* 1. API library (Strconv)

cheeseId, err := strconv.ParseInt(recordSlice[0], 10, 64)

if err != nil { cheeseId = 0 }

fatContentPercent, err := strconv.ParseFloat(recordSlice[6], 32)

if err != nil { fatContentPercent = 0.0 }

moisturePercent, err := strconv.ParseFloat(recordSlice[7], 32)

if err != nil { moisturePercent = 0.0 }

organic, err := strconv.ParseBool(recordSlice[12])

if err != nil { organic = false }

* 1. Array (called a Slice in Go)

var recordSlice []string

recordSlice = append(recordSlice, readString("Cheese ID (int)")) //0

recordSlice = append(recordSlice, readString("Cheese Name")) //1

* 1. Unit Testing

func TestLoadData(t \*testing.T) {

    // create a record with the proper data

    firstRecord := Record {

        CheeseId: 228,

        CheeseName: "Sieur de Duplessis (Le)",

        ManufacturerName: "Fromages la faim de loup",

        ManufacturerProvCode: "NB",

        ManufacturingType: "Farmstead",

        WebSite: "N/A",

        FatContentPercent: 24.2,

        MoisturePercent: 47,

        Particularities: "N/A",

        Flavour: "Sharp, lactic",

        Characteristics: "Uncooked",

        Ripening: "9 Months",

        Organic: false,

        CategoryType: "Firm Cheese",

        MilkType: "Ewe",

        MilkTreatmentType: "Raw Milk",

        RindType: "Washed Rind",

        LastUpdateDate: "2016-02-03",

    }

    // load data and get the first record

    records := loadData("data/canadianCheeseDirectory.csv", 5)

    loadedFirstRecord := records[0]

    // check if loaded first record and our test record are equal

    if !reflect.DeepEqual(firstRecord, loadedFirstRecord) {

       t.Errorf("Loaded First Record was incorrect, \n got: \n%+v\n, want: \n%+v\n", firstRecord, loadedFirstRecord)

    }

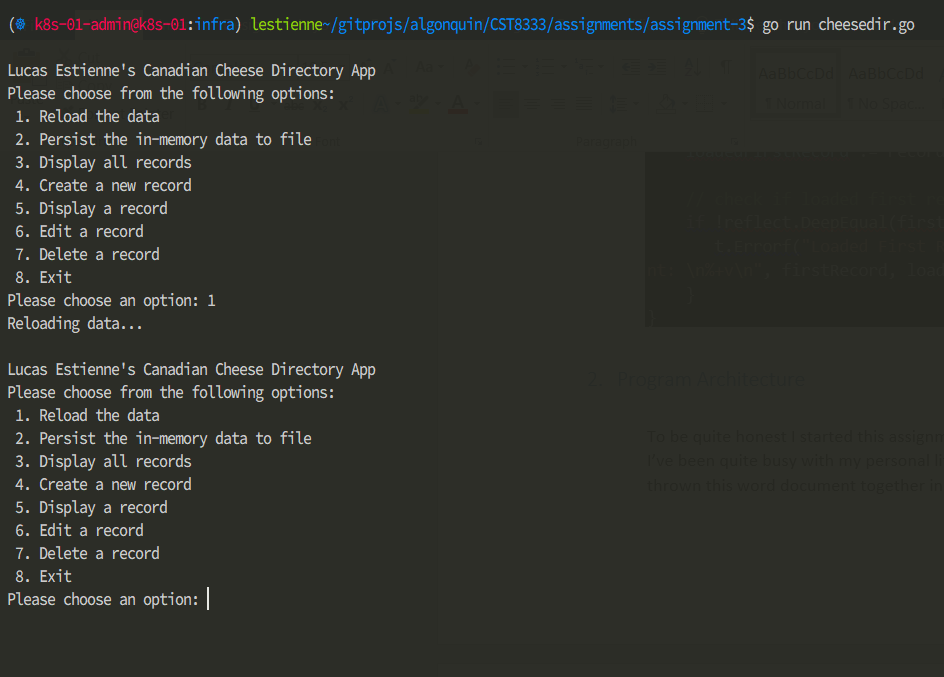
}

## Program Architecture

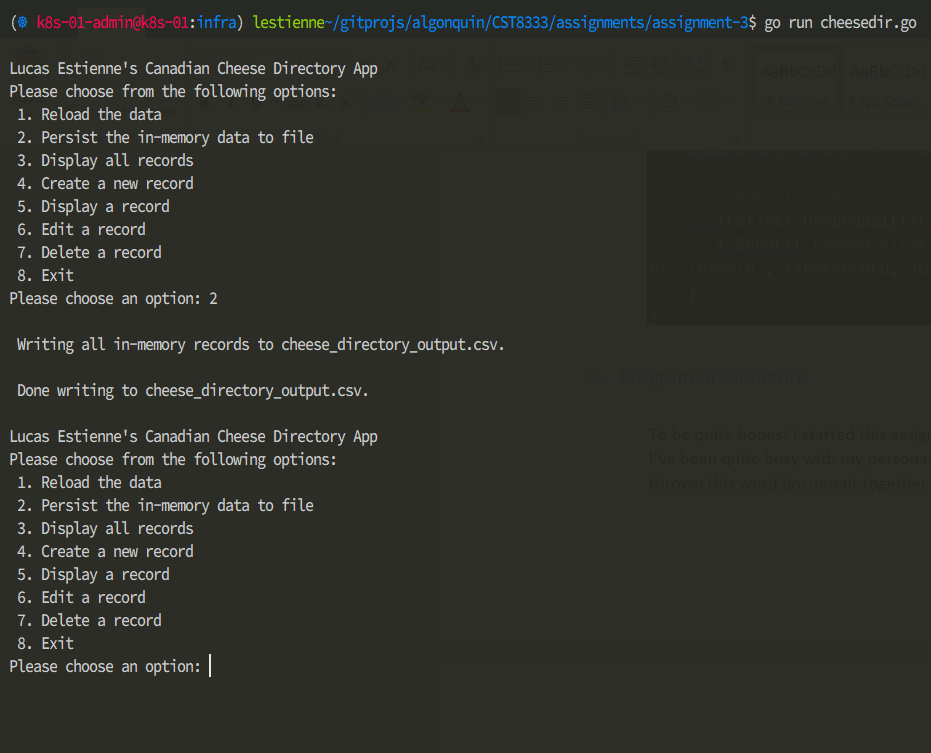
To be quite honest I started this assignment at around 8:30PM on the deadline day since I’ve been quite busy with my personal life so I haven’t had time to complete this section. I’ve thrown this word document together in about 15 minutes, submitting in the last 10 min before the deadline.

## Program Demonstration via Screen Shot

Reloading:



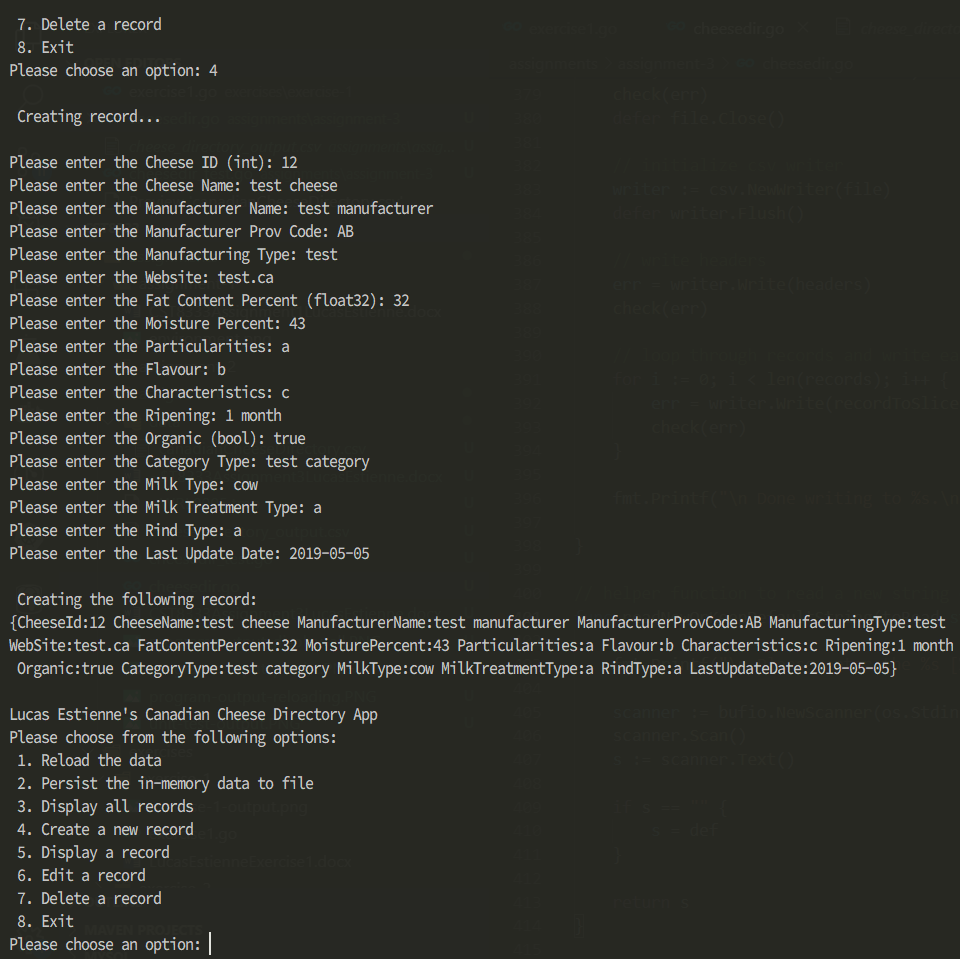
Persist:



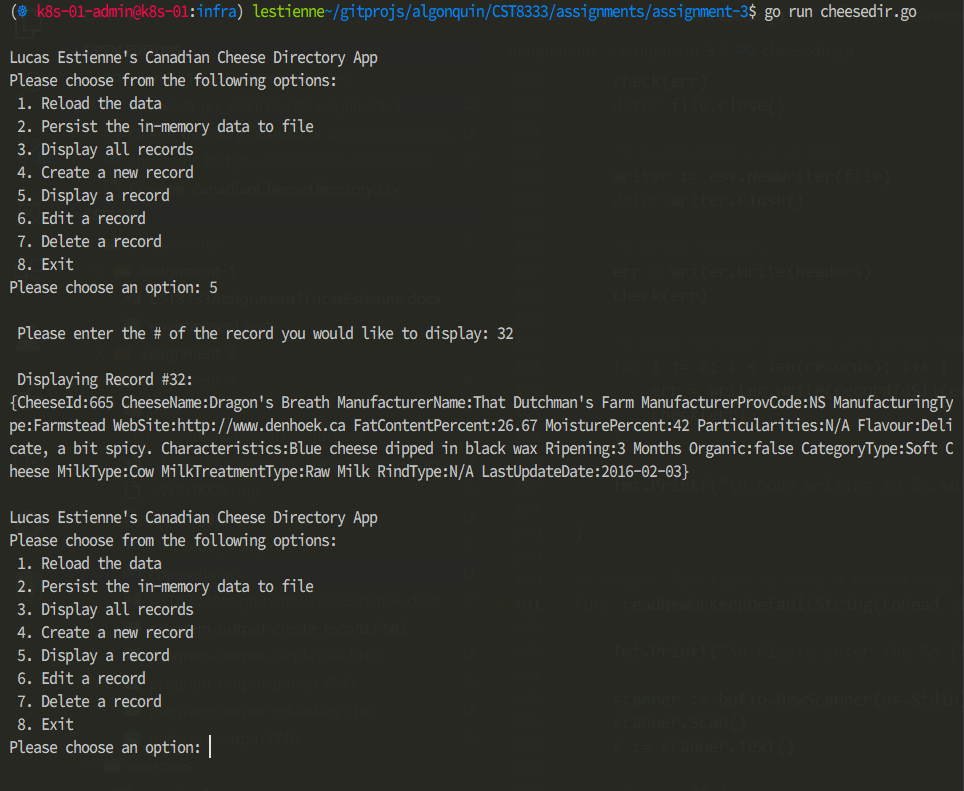
Display All:



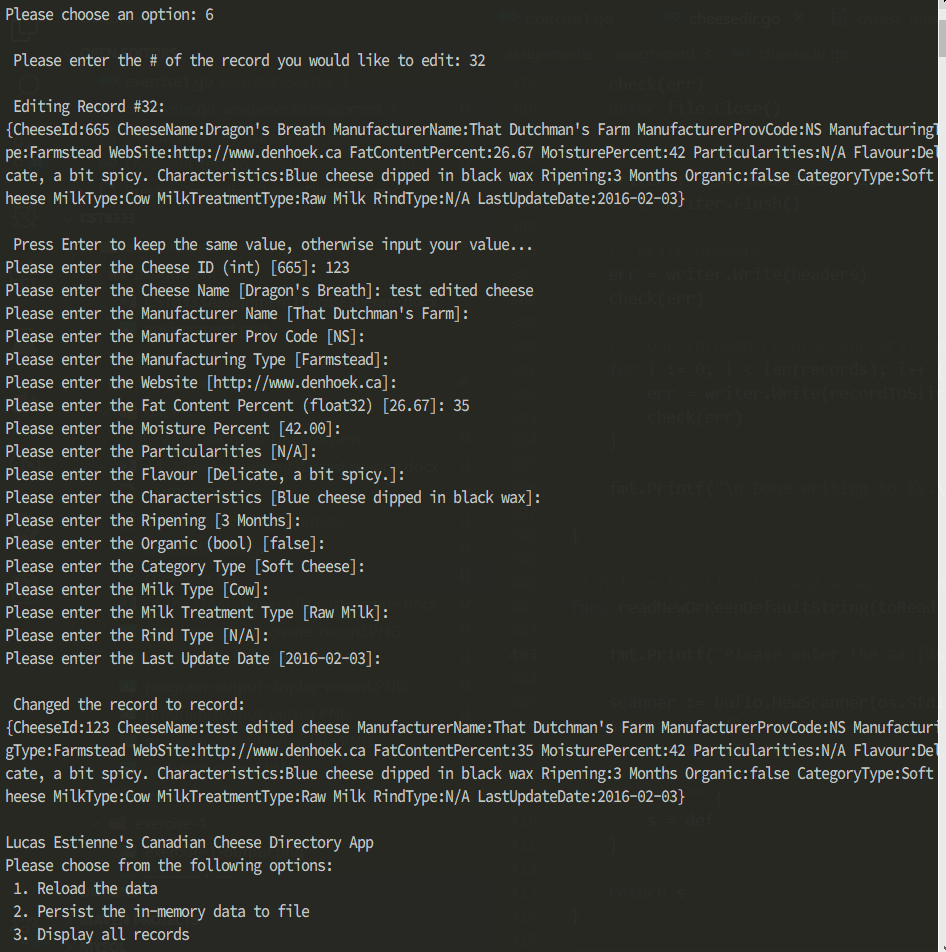
Create new record:



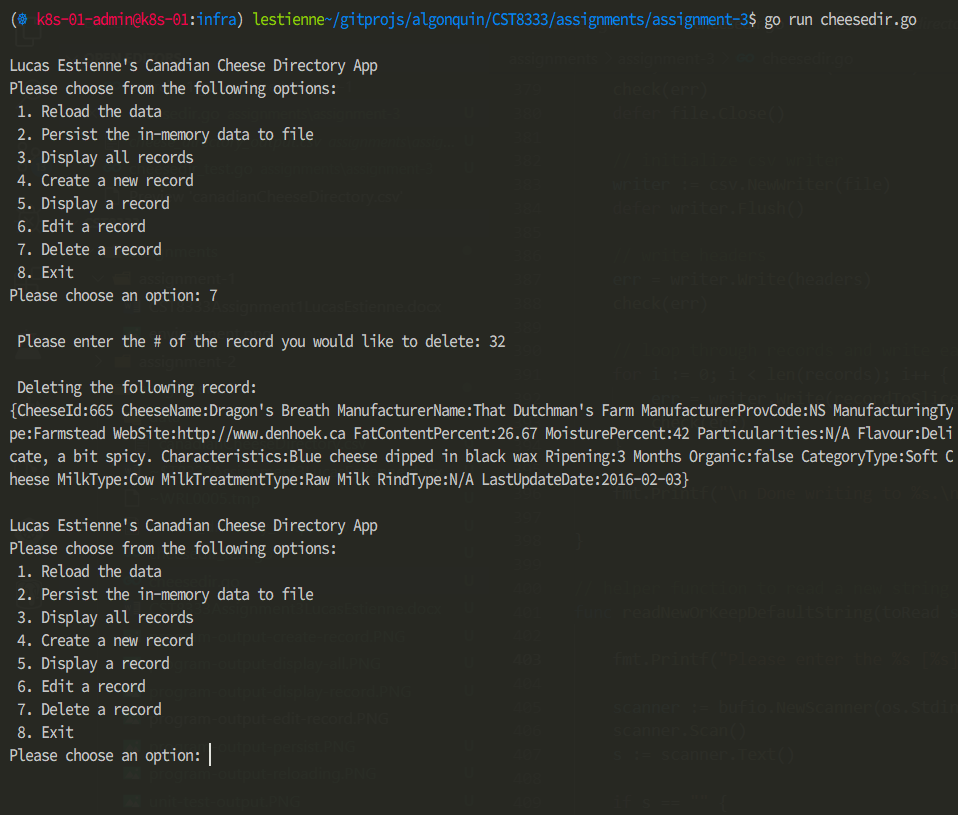
Display a record:



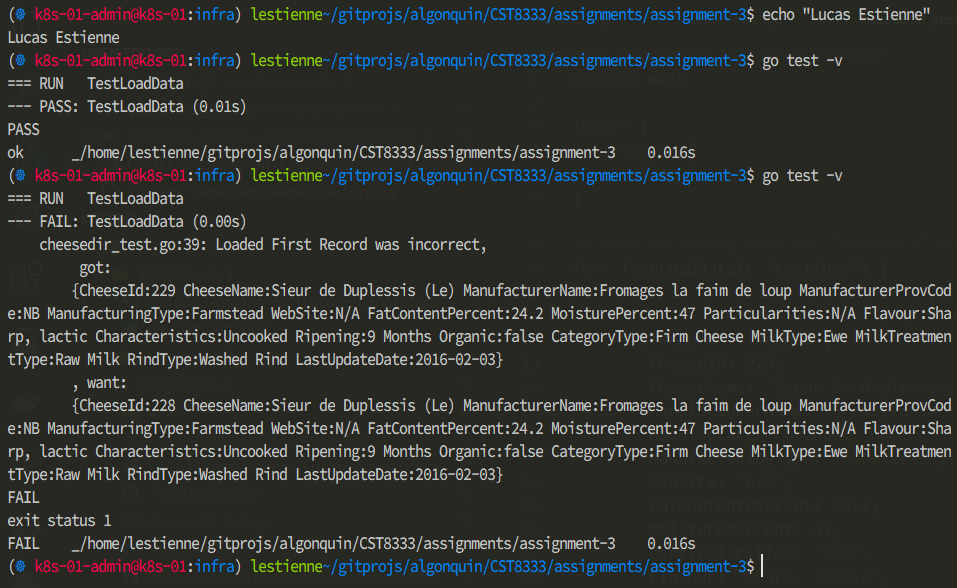
Edit record:



Delete record:



## Unit Testing Demonstration via Screen Shots



## Source Code

### cheesedir.go

// CST8333 Cheese Directory App - Lucas Estienne

package main

import (

    "fmt"

    "os"

    "log"

    "time"

    "bufio"

    "strings"

    "strconv"

    "encoding/csv"

)

const NumRecordsToLoad = 10000

const (

    OptionReload = 1

    OptionPersist = 2

    OptionDisplayAll = 3

    OptionCreate = 4

    OptionDisplay = 5

    OptionEdit = 6

    OptionDelete = 7

    OptionExit = 8

)

// simple data structure containing a string

type Record struct {

    CheeseId int

    CheeseName string

    ManufacturerName string

    ManufacturerProvCode string

    ManufacturingType string

    WebSite string

    FatContentPercent float32

    MoisturePercent float32

    Particularities string

    Flavour string

    Characteristics string

    Ripening string

    Organic bool

    CategoryType string

    MilkType string

    MilkTreatmentType string

    RindType string

    LastUpdateDate string

}

// main function, this is the entrypoint

func main() {

    // load data

    records := loadData("data/canadianCheeseDirectory.csv", NumRecordsToLoad)

    // loop until exit

    for true {

        // display menu and process choice

        switch selection := showMenu(); selection {

            case OptionReload:

                fmt.Println("Reloading data...")

                records = loadData("data/canadianCheeseDirectory.csv", NumRecordsToLoad)

            case OptionPersist:

                persistToFile(records, "cheese\_directory\_output.csv")

            case OptionDisplayAll:

                displayAllRecords(records)

            case OptionCreate:

                records = createRecord(records)

            case OptionDisplay:

                displayRecord(records)

            case OptionEdit:

                editRecord(records)

            case OptionDelete:

                records = deleteRecord(records)

            case OptionExit:

                fmt.Println("Goodbye")

                return

        }

        time.Sleep(1 \* time.Second)

    }

}

// helper function to do error handling

func check(e error) {

    if e != nil {

        log.Fatal("Error", e)

        panic(e)

    }

}

// helper function to return the first of two non empty strings, or the string "N/A"

func getFirstNonEmptyStringOrNA(first string, second string) string {

    if strings.TrimSpace(first) != "" {

        return first

    } else if strings.TrimSpace(second) != "" {

        return second

    } else {

        return "N/A"

    }

}

// helper function to read CSV

func getLinesFromCSV(filePath string) (lines [][]string, err error) {

    // open file

    file, err := os.Open(filePath)

    check(err)

    defer file.Close() // defer closing the file until function returns

    // create CSV Reader from file

    reader := csv.NewReader(file)

    return reader.ReadAll()

}

// function to convert CSV line to Record object

func lineToRecord(line []string) Record {

    // parse some values from strings

    cheeseId, err := strconv.ParseInt(line[0], 10, 64)

    if err != nil { cheeseId = 0 }

    fatContentPercent, err := strconv.ParseFloat(line[10], 32)

    if err != nil { fatContentPercent = 0.0 }

    moisturePercent, err := strconv.ParseFloat(line[11], 32)

    if err != nil { moisturePercent = 0.0 }

    organic, err := strconv.ParseBool(line[20])

    if err != nil { organic = false }

    return Record {

        CheeseId: int(cheeseId),

        CheeseName: getFirstNonEmptyStringOrNA(line[1], line[2]),

        ManufacturerName: getFirstNonEmptyStringOrNA(line[3], line[4]),

        ManufacturerProvCode: getFirstNonEmptyStringOrNA(line[5], "??"),

        ManufacturingType: getFirstNonEmptyStringOrNA(line[6], line[7]),

        WebSite: getFirstNonEmptyStringOrNA(line[8], line[9]),

        FatContentPercent: float32(fatContentPercent),

        MoisturePercent: float32(moisturePercent),

        Particularities: getFirstNonEmptyStringOrNA(line[12], line[13]),

        Flavour: getFirstNonEmptyStringOrNA(line[14], line[15]),

        Characteristics: getFirstNonEmptyStringOrNA(line[16], line[17]),

        Ripening: getFirstNonEmptyStringOrNA(line[18], line[19]),

        Organic: organic,

        CategoryType: getFirstNonEmptyStringOrNA(line[21], line[22]),

        MilkType: getFirstNonEmptyStringOrNA(line[23], line[24]),

        MilkTreatmentType: getFirstNonEmptyStringOrNA(line[25], line[26]),

        RindType: getFirstNonEmptyStringOrNA(line[27], line[28]),

        LastUpdateDate: line[29],

    }

}

// function load or reload data

func loadData(filePath string, numRecords int) []Record {

    var records []Record

    // Load lines from CSV

    lines, err := getLinesFromCSV(filePath)

    check(err)

    // get rid of column names

    lines = lines[1:]

    // convert lines to records slice

    for i := 0; i < numRecords && i < len(lines); i++ {

        records = append(records, lineToRecord(lines[i]))

    }

    return records

}

// function to show menu and return the user selection

func showMenu() int {

    selection := 0

    fmt.Println("\nLucas Estienne's Canadian Cheese Directory App")

    fmt.Println("Please choose from the following options:")

    fmt.Println(" 1. Reload the data")

    fmt.Println(" 2. Persist the in-memory data to file")

    fmt.Println(" 3. Display all records")

    fmt.Println(" 4. Create a new record")

    fmt.Println(" 5. Display a record")

    fmt.Println(" 6. Edit a record")

    fmt.Println(" 7. Delete a record")

    fmt.Println(" 8. Exit")

    // loop until selection is valid

    for selection == 0 {

        fmt.Printf("Please choose an option: ")

        \_, err := fmt.Scanf("%d", &selection)

        if err != nil {

            selection = 0

            fmt.Println("\nPlease enter a valid option.")

        } else if selection < 1 || selection > 8 {

            selection = 0

            fmt.Println("\nPlease enter a valid integer between 1 and 8.")

        }

    }

    return selection

}

// function to display all records

func displayAllRecords(records []Record) {

    fmt.Printf("\nDisplaying all records...\n\n")

    for i := 0; i < len(records); i++ {

        fmt.Printf("Record ID: %d: %+v\n", i, records[i])

        time.Sleep(5 \* time.Millisecond) // 5ms between records

    }

}

// function to display a specific record

func displayRecord(records []Record) {

    id := -1

    // loop until ID is valid

    for id == -1 {

        fmt.Printf("\n Please enter the # of the record you would like to display: ")

        \_, err := fmt.Scanf("%d", &id)

        if err != nil {

            id = -1

            fmt.Println("\nPlease enter a valid integer.")

        } else if id < 0 || id > len(records)-1 {

            id = -1

            fmt.Printf("\nPlease enter a valid record ID between 0 and %d.\n", len(records)-1)

        }

    }

    // display record

    fmt.Printf("\n Displaying Record #%d: \n%+v\n", id, records[id])

}

// helper function to delete an element from a Record slice and keep order

func deleteRecordFromSlice(slice []Record, id int) []Record {

    return append(slice[:id], slice[id+1:]...)

}

func deleteRecord(records []Record) []Record {

    id := -1

    // loop until ID is valid

    for id == -1 {

        fmt.Printf("\n Please enter the # of the record you would like to delete: ")

        \_, err := fmt.Scanf("%d", &id)

        if err != nil {

            id = -1

            fmt.Println("\nPlease enter a valid integer.")

        } else if id < 0 || id > len(records)-1 {

            id = -1

            fmt.Printf("\nPlease enter a valid record ID between 0 and %d.\n", len(records)-1)

        }

    }

    // display the record we are deleting

    fmt.Printf("\n Deleting the following record: \n%+v\n", records[id])

    // return a slice with the element removed

    return deleteRecordFromSlice(records, id)

}

// helper function to read a string from stdin

func readString(toRead string) string {

    fmt.Printf("Please enter the %s: ", toRead)

    // read from scanner

    scanner := bufio.NewScanner(os.Stdin)

    scanner.Scan()

    s := scanner.Text()

    if s == "" {

        s = "N/A"

    }

    return s

}

func createRecord(records []Record) []Record {

    var recordSlice []string

    fmt.Printf("\n Creating record...\n\n")

    // read values for our record

    recordSlice = append(recordSlice, readString("Cheese ID (int)")) //0

    recordSlice = append(recordSlice, readString("Cheese Name")) //1

    recordSlice = append(recordSlice, readString("Manufacturer Name")) //2

    recordSlice = append(recordSlice, readString("Manufacturer Prov Code")) //3

    recordSlice = append(recordSlice, readString("Manufacturing Type")) //4

    recordSlice = append(recordSlice, readString("Website")) //5

    recordSlice = append(recordSlice, readString("Fat Content Percent (float32)")) //6

    recordSlice = append(recordSlice, readString("Moisture Percent")) //7

    recordSlice = append(recordSlice, readString("Particularities")) //8

    recordSlice = append(recordSlice, readString("Flavour")) //9

    recordSlice = append(recordSlice, readString("Characteristics")) //10

    recordSlice = append(recordSlice, readString("Ripening")) //11

    recordSlice = append(recordSlice, readString("Organic (bool)")) //12

    recordSlice = append(recordSlice, readString("Category Type")) //13

    recordSlice = append(recordSlice, readString("Milk Type")) //14

    recordSlice = append(recordSlice, readString("Milk Treatment Type")) //15

    recordSlice = append(recordSlice, readString("Rind Type")) //16

    recordSlice = append(recordSlice, readString("Last Update Date")) //17

    // parse some values from strings

    cheeseId, err := strconv.ParseInt(recordSlice[0], 10, 64)

    if err != nil { cheeseId = 0 }

    fatContentPercent, err := strconv.ParseFloat(recordSlice[6], 32)

    if err != nil { fatContentPercent = 0.0 }

    moisturePercent, err := strconv.ParseFloat(recordSlice[7], 32)

    if err != nil { moisturePercent = 0.0 }

    organic, err := strconv.ParseBool(recordSlice[12])

    if err != nil { organic = false }

    // init record

    r := Record {

        CheeseId: int(cheeseId),

        CheeseName: recordSlice[1],

        ManufacturerName: recordSlice[2],

        ManufacturerProvCode: recordSlice[3],

        ManufacturingType: recordSlice[4],

        WebSite: recordSlice[5],

        FatContentPercent: float32(fatContentPercent),

        MoisturePercent: float32(moisturePercent),

        Particularities: recordSlice[8],

        Flavour: recordSlice[9],

        Characteristics: recordSlice[10],

        Ripening: recordSlice[11],

        Organic: organic,

        CategoryType: recordSlice[13],

        MilkType: recordSlice[14],

        MilkTreatmentType: recordSlice[15],

        RindType: recordSlice[16],

        LastUpdateDate: recordSlice[17],

    }

    fmt.Printf("\n Creating the following record: \n%+v\n", r)

    // return our records slice with the new record appended

    return append(records, r)

}

// helper function to convert a Record object to a slice

func recordToSlice(record Record) []string {

    var recordSlice []string

    recordSlice = []string{

        fmt.Sprintf("%d",record.CheeseId), record.CheeseName, record.ManufacturerName, record.ManufacturerProvCode,

        record.ManufacturingType, record.WebSite, fmt.Sprintf("%.2f", record.FatContentPercent),

        fmt.Sprintf("%.2f", record.MoisturePercent), record.Particularities, record.Flavour,

        record.Characteristics, record.Ripening, fmt.Sprintf("%t", record.Organic),

        record.CategoryType, record.MilkType, record.MilkTreatmentType, record.RindType, record.LastUpdateDate,

    }

    return recordSlice

}

// function to write in-memory records to file

func persistToFile(records []Record, filePath string) {

    fmt.Printf("\n Writing all in-memory records to %s.\n", filePath)

    headers :=  []string {

        "CheeseId", "CheeseName", "ManufacturerName", "ManufacturerProvCode", "ManufacturingType",

        "WebSite", "FatContentPercent", "MoisturePercent", "Particularities", "Flavour", "Characteristics",

        "Ripening", "Organic", "CategoryType", "MilkType", "MilkTreatmentType", "RindType", "LastUpdateDate",

    }

    // create file

    file, err := os.Create(filePath)

    check(err)

    defer file.Close()

    // initialize csv writer

    writer := csv.NewWriter(file)

    defer writer.Flush()

    // write headers

    err = writer.Write(headers)

    check(err)

    // loop through records and write each one to the CSV

    for i := 0; i < len(records); i++ {

        err = writer.Write(recordToSlice(records[i]))

        check(err)

    }

    fmt.Printf("\n Done writing to %s.\n", filePath)

}

// helper function to read a new string or keep the provided default

func readNewOrKeepDefaultString(toRead string, def string) string {

    fmt.Printf("Please enter the %s [%s]: ", toRead, def)

    // read from scanner

    scanner := bufio.NewScanner(os.Stdin)

    scanner.Scan()

    s := scanner.Text()

    if s == "" {

        s = def

    }

    return s

}

// function to edit record

func editRecord(records []Record) []Record {

    var recordSlice []string

    id := -1

    // loop until ID is valid

    for id == -1 {

        fmt.Printf("\n Please enter the # of the record you would like to edit: ")

        \_, err := fmt.Scanf("%d", &id)

        if err != nil {

            id = -1

            fmt.Println("\nPlease enter a valid integer.")

        } else if id < 0 || id > len(records)-1 {

            id = -1

            fmt.Printf("\nPlease enter a valid record ID between 0 and %d.\n", len(records)-1)

        }

    }

    r := records[id]

    // edit record

    fmt.Printf("\n Editing Record #%d: \n%+v\n", id, r)

    fmt.Printf("\n Press Enter to keep the same value, otherwise input your value...\n")

    // read values for our record

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Cheese ID (int)", fmt.Sprintf("%d",r.CheeseId))) //0

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Cheese Name", r.CheeseName)) //1

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Manufacturer Name", r.ManufacturerName)) //2

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Manufacturer Prov Code", r.ManufacturerProvCode)) //3

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Manufacturing Type", r.ManufacturingType)) //4

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Website", r.WebSite)) //5

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Fat Content Percent (float32)", fmt.Sprintf("%.2f",r.FatContentPercent))) //6

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Moisture Percent", fmt.Sprintf("%.2f",r.MoisturePercent))) //7

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Particularities", r.Particularities)) //8

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Flavour", r.Flavour)) //9

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Characteristics", r.Characteristics)) //10

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Ripening", r.Ripening)) //11

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Organic (bool)", fmt.Sprintf("%t",r.Organic))) //12

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Category Type", r.CategoryType)) //13

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Milk Type", r.MilkType)) //14

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Milk Treatment Type", r.MilkTreatmentType)) //15

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Rind Type", r.RindType)) //16

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Last Update Date", r.LastUpdateDate)) //17

    // parse some values from strings

    cheeseId, err := strconv.ParseInt(recordSlice[0], 10, 64)

    if err != nil { cheeseId = 0 }

    fatContentPercent, err := strconv.ParseFloat(recordSlice[6], 32)

    if err != nil { fatContentPercent = 0.0 }

    moisturePercent, err := strconv.ParseFloat(recordSlice[7], 32)

    if err != nil { moisturePercent = 0.0 }

    organic, err := strconv.ParseBool(recordSlice[12])

    if err != nil { organic = false }

    // replace record

    records[id] = Record {

        CheeseId: int(cheeseId),

        CheeseName: recordSlice[1],

        ManufacturerName: recordSlice[2],

        ManufacturerProvCode: recordSlice[3],

        ManufacturingType: recordSlice[4],

        WebSite: recordSlice[5],

        FatContentPercent: float32(fatContentPercent),

        MoisturePercent: float32(moisturePercent),

        Particularities: recordSlice[8],

        Flavour: recordSlice[9],

        Characteristics: recordSlice[10],

        Ripening: recordSlice[11],

        Organic: organic,

        CategoryType: recordSlice[13],

        MilkType: recordSlice[14],

        MilkTreatmentType: recordSlice[15],

        RindType: recordSlice[16],

        LastUpdateDate: recordSlice[17],

    }

    fmt.Printf("\n Changed the record to record: \n%+v\n", records[id])

    // return our amended records slice

    return records

}

### cheesedir\_test.go

// CST8333 Cheese Directory App - Unit Tests - Lucas Estienne

package main

import (

    "testing"

    "reflect"

)

// test to verify that our "loadData" function loads the first record from the dataset properly

func TestLoadData(t \*testing.T) {

    // create a record with the proper data

    firstRecord := Record {

        CheeseId: 229,

        CheeseName: "Sieur de Duplessis (Le)",

        ManufacturerName: "Fromages la faim de loup",

        ManufacturerProvCode: "NB",

        ManufacturingType: "Farmstead",

        WebSite: "N/A",

        FatContentPercent: 24.2,

        MoisturePercent: 47,

        Particularities: "N/A",

        Flavour: "Sharp, lactic",

        Characteristics: "Uncooked",

        Ripening: "9 Months",

        Organic: false,

        CategoryType: "Firm Cheese",

        MilkType: "Ewe",

        MilkTreatmentType: "Raw Milk",

        RindType: "Washed Rind",

        LastUpdateDate: "2016-02-03",

    }

    // load data and get the first record

    records := loadData("data/canadianCheeseDirectory.csv", 5)

    loadedFirstRecord := records[0]

    // check if loaded first record and our test record are equal

    if !reflect.DeepEqual(firstRecord, loadedFirstRecord) {

       t.Errorf("Loaded First Record was incorrect, \n got: \n%+v\n, want: \n%+v\n", firstRecord, loadedFirstRecord)

    }

}