CS23710 - Assignment 1: Relocate Aberyswyth Dairy Company

Alexander Brown

December 16, 2010

Contents

1 Design

1.1 The Problem

From the way the assignment is presented, there are two problems to solve - minimizing the distance and minimizing the cost.

I decided it would be best to seperate the program into these two problems. The distance I chose to name "Simple Relocate" and the cost I chose to name "Complex Relocate", mainly due to the way the problem is presented.

Also I decided on the use of command line arguments. The way to run the simple or complex relocate would be switched by these command line arguments, along with version and help information.

1.1.1 Arguments

With no arguments, the relocate program defaults to telling the user that there is a missing file operand and to use `relocate -h` to view the help information.

With only a file operand the relocate program defaults to the Complex Relocate program.

With options that don't exist the relocate program tells the user that there is no such option and to use `relocate -h` to view the help information.

Printing Help

```
relocate -h
relocate --help
```

Version Information

```
relocate -v
relocate --version
```

Simple Relocate

```
relocate -s file_path
relocate --simple file_path
```

Complex Relocate

```
relocate -c file_path
relocate --complex file_path
```

1.2 Handelling the Data Files

The obvious solution to handelling the data files is through the use of structures.

1.2.1 The Company Description File

The Company Description File is simple a file which points the program to the files which contain the actual information. It should read the first file, load the necessary information from that file, then move onto the next file and continue until it's reached the end of the company description file.

However due to the nature of how a company works I thought it best to also have a company structure which can hold all the information needed on a company in a single structure, rather than having to keep them in method variables.

1.2.2 The Region File

The first line of this file is merely a string to be read in.

It is then followed by a list of locations. These I have decided it will be best to store in a seperate structure, which is then stored as an array in the region structure.

region

name The name of the region.

locations The locations in this region.

location

code The location code number.

name The name of the location.

1.2.3 The Roads File

This file handles the data for the roads, for which a structure has already been defined. It is a simple procedure of reading in the roads as a linked list.

1.2.4 The Business Park File

This is the only file which doesn't require a structure as the data can simply be stored in an array of intergers.

1.2.5 The Deliveries File

The deliveries file handles all the deliveries that have to be made. Again this is best stored as a structure.

deliveries

retailer The code for the retailer.

location The location code of the retailer.

deliveries The number of deliveries per week.

weight The weight of the delivery.

1.2.6 The Running Costs File

This file is only necessary for the complex relocate program, however I had it load into the company file so the information is easily available.

1.3 Program structure

Being used to Object Orientated Programming, I took a very similar approach and segmented my program into as many parts as was possible. Whilst most files only contain one or two methods, I prefer to have such seperation to make the code both easier to read and to modify.

I also made use of header files, holding the structures and method initializers in these header files as is typically done. Then the source files which manipulate these structures, finally a single source file to run the program holding the main method. I also used a miscellaneous library (misc.h and misc.c) to hold general methods that can be used over many files.

2 Further Documentation

I have used Doxygen to create in-depth documentation (hence the unusual style of comments). If you wish to view any further documentation please look at the LaTeX output and HTML files found in the `documentation` directory.

3 Review

I found this project fairly easy to work with, as the vast majority was just structure manipulation. There is one small bug in that the program cannot yet handle relative filenames. I could solve this by either using strcat, or by using macros, but did not have time to do so.

4 C Code

4.1 relocate.c

```
/**
 * @brief The main class for the relocating program.
 * @file relocate.c
 * @author Alexander Brown
 *
 * Created 14 November 2010
 */

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

#include "complexRelocate.h"
#include "simpleRelocate.h"

/**
 * @brief The current version of the program.
 */
#define VERSION "1.0"

/**
 * @brief The path to the man page.
 */
#define HELP_FILE "man/relocate.txt"

#define SIMPLE_RELOCATE (1)
#define COMPLEX_RELOCATE (2)

/**
 * @brief Prints the man page when the option '-h' or '--help' is used.
 * @see VERSION
 */
void printHelp() {
  FILE *helpFile = fopen(HELP_FILE,"r");
  if(NULL==helpFile) {
    printf("Help not found\n");
  }
  else {
    int c;
    while((c=fgetc(helpFile))!=EOF) {
      printf("%c",c);
    }
    fclose(helpFile);
  }
}

/**
 * @brief Runs the relocate program
 * @param argc The number of arguments specified.
 * @param argv The arguments specified.
 * @return The typical C return type depending on the success or failure of the program.
 */
int main(int argc, char **argv) {
  if(argc < 2) {
    printf("relocate: missing file operand\nTry 'relocate --help' for more information.\n");
  }
  else {
    FILE *file = NULL;
    int runningOption = COMPLEX_RELOCATE;

    int i;
    for(i=1;i<argc;i++) {
      if(strcmp(argv[i], "-h")==0 || strcmp(argv[i], "--help")==0) {
        printHelp();
        return EXIT_SUCCESS;
      }
      else if(strcmp(argv[i], "-v")==0 || strcmp(argv[i], "--version")==0) {
        printf("Relocating Aberystwyth Dairy Company program by Alex Brown, Version %s\n\n",VERSION);
        return EXIT_SUCCESS;
      }
      else if(strcmp(argv[i], "-s")==0 || strcmp(argv[i], "--simple")==0) {
```

```
      runningOption = SIMPLE_RELOCATE;
    }
    else if(strcmp(argv[i], "-c")==0 || strcmp(argv[i], "--complex")==0) {
      runningOption = COMPLEX_RELOCATE;
    }
    else if(strncmp(argv[i], "-",1)!=0) { /* If there isn't an option - assume it's a file */
      file = fopen(argv[i],"r");
      if(0==file) {
        printf("Unrecognised file\n");
        return EXIT_FAILURE;
      }
    }
    else {
      printf("relocate: unrecognised option '%s'\nTry `relocate --help' for more information.\n",argv[i]);
      return EXIT_FAILURE;
    }
  }

  if(NULL != file) {
    if(SIMPLE_RELOCATE == runningOption)
      runSimple(file);
    else if(COMPLEX_RELOCATE == runningOption)
      runComplex(file);

    fclose(file);
  }
  else {
    printf("relocate: file not specified\nTry `relocate --help' for more information.\n");
    return EXIT_FAILURE;
  }
 }

 return EXIT_SUCCESS;
}
```

4.2 simpleRelocate.h

```
/*
 * File:   simpleRelocate.h
 * Author: adb9
 *
 * Created 14 November 2010
 */

#include <stdio.h>

#include "company.h"

#ifndef _SIMPLE_RELOCATE
#define _SIMPLE_RELOCATE

void runSimple(FILE * mainFile);

int getDistanceFromPark(int location, company company);

#endif
```

4.3 simpleRelocate.c

```
/**
 * @brief The source file for the simple run routine.
 * @file
 * @author Alexander Brown
 *
 * Created 14 November 2010
 *
 * Modified 09 December 2010 - Changed to use NULL instead of 0.
 */

#include <stdlib.h>
#include <stdio.h>

#include "simpleRelocate.h"
#include "company.h"
```

```
#include "region.h"
#include "delivery.h"
#include "../lib/shortest.h"

void runSimple(FILE *file) {
  company company;
  int i;
  int *distances;
  int shortestPath;
  int bestLocation;

  loadCompany(file, &company);

  printf("Company name: %s\n",company.name);

  distances = calloc(company.numberOfParks,sizeof(int));

  distances[0] = getDistanceFromPark(company.parks[0],company);
  shortestPath = distances[0];
  bestLocation = company.parks[0];

  for(i=1;i<company.numberOfParks;i++) {
    distances[i] = getDistanceFromPark(company.parks[i],company);
    if(distances[i] < shortestPath) {
      shortestPath = distances[i];
      bestLocation = company.parks[i];
    }
  }

  free(distances);
  /* Free up memory */
  for(i=0;i<company.region.numberOfLocations;i++)
    free(company.region.locations[i].name);
  free(company.region.locations);
  free(company.region.name);
  free(company.name);

  while(NULL != company.roads->next) {
    road *temp = company.roads->next;
        free(company.roads);
        company.roads = temp;
  }

  printf("Best location is %s (with a total distance of %d miles).\n",
    company.region.locations[bestLocation].name,
    shortestPath);
}

int getDistanceFromPark(int location, company company) {
  int distance = 0;
  int i;
  road *route;

  for(i=0;i<company.numberOfDeliveries;i++) {
    int currentDistance = 0;

    route = shortest_route(location,
      company.deliveries[i].location,
      company.region.numberOfLocations,
      company.roads);

    while(NULL != route) {
      currentDistance += route->length;
      route = route->next;
    }

    currentDistance *= company.deliveries[i].deliveries;
    distance += currentDistance;
  }

  return distance*2; /* Times by two for there and back */
}
```

4.4 complexRelocate.h

```
/**
 * @brief The header file for the complex run routine.
 * @file
 * @author Alexander Brown
 *
 * Created 14 November 2010
 */

#include <stdio.h>

#include "company.h"

#ifndef _COMPLEX_RELOCATE
#define _COMPLEX_RELOCATE

/**
 * @brief Runs the program using the cost of making deliveries.
 *
 * @param mainFile The input file from the user.
 */
void runComplex(FILE * mainFile);

/**
 * @brief Gets the cost from a particular location using the details of a particular company.
 *
 * @param location The reference number of the park @see region.locations
 * @param company The company to relocate.
 */
float getCostFromPark(int location, company company);

#endif
```

4.5 complexRelocate.c

```
/**
 * @brief The source file for the complex run routine.
 * @file
 * @author Alexander Brown
 *
 * Created 14 November 2010
 *
 * Modified 09 December 2010 - Changed to use NULL instead of 0.
 */

#include <stdlib.h>
#include <stdio.h>

#include "../lib/shortest.h"

#include "company.h"
#include "complexRelocate.h"
#include "delivery.h"
#include "misc.h"
#include "region.h"

void runComplex(FILE *file) {
  company company;
  int i;
  float *costs;
  float lowestCost;
  int bestLocation;

  loadCompany(file, &company);

  printf("Company name: %s\n",company.name);

  costs = calloc(company.numberOfParks,sizeof(float));

  costs[0] = getCostFromPark(company.parks[0],company);
  lowestCost = costs[0];
  bestLocation = company.parks[0];

  for(i=1;i<company.numberOfParks;i++) {
    costs[i] = getCostFromPark(company.parks[i],company);
```

```
    if(costs[i] < lowestCost) {
      lowestCost = costs[i];
      bestLocation = company.parks[i];
    }
  }

  free(costs);
  /* Free up memory */
  for(i=0;i<company.region.numberOfLocations;i++)
    free(company.region.locations[i].name);
  free(company.region.locations);
  free(company.region.name);
  free(company.name);

  while(NULL != company.roads->next) {
    road *temp = company.roads->next;
        free(company.roads);
        company.roads = temp;
  }

  lowestCost /= 100; /* Divide by 100 to get pounds from pence */

  printf("Best location is %s (with a running cost of  %.2f).\n",
    company.region.locations[bestLocation].name,
    lowestCost);

}

float getCostFromPark(int location, company company) {
  float cost = 0;
  int i;
  road *route;

  for(i=0;i<company.numberOfDeliveries;i++) {
    int currentCost = 0;

    route = shortest_route(location,
      company.deliveries[i].location,
      company.region.numberOfLocations,
      company.roads);

    while(NULL != route) {
      currentCost += route->length;
      route = route->next;
    }

    currentCost *= company.deliveries[i].deliveries;
    cost += currentCost * (company.runningCostEmpty +
      (company.runningCostPerKg * company.deliveries[i].weight));
    cost += currentCost * company.runningCostEmpty;
  }

  return cost;
}
```

4.6 company.h

```
/**
 * @brief The header file for a company.
 * @file company.h
 * @author Alexander Brown
 *
 * Created 14 November 2010
 */

#include "region.h"
#include "delivery.h"
#include "../lib/shortest.h"

#ifndef MAX_NAME_SIZE
/** The maximum size of any name string for a region or locations. */
#define MAX_NAME_SIZE 255
#endif

#ifndef _COMPANY
```

```
#define _COMPANY

/**
 * @brief Structure for a company
 */
typedef struct {
  /** The name of the company. */
  char *name;

  /** The region which this company is located in. */
  region region;

  /** The linked list of roads. */
  road *roads;

  /** The parks in the region. */
  int *parks;

  /** The number of parks in the region. */
  int numberOfParks;

  /** The deliveries of the company. */
  delivery *deliveries;

  /**
   * The number of deliveries.
   */
  int numberOfDeliveries;

  /**  The cost of running a van empty. */
  float runningCostEmpty;

  /** The extra cost per kilogram of running a van. */
  float runningCostPerKg;
} company;

/**
 * @brief Loads a company from it's relevant description file.
 *
 * @param file The company description file.
 * @param company The company to set up.
 */
void loadCompany(FILE * file, company * company);

/**
 * @brief Loads the roads of a company.
 * @param file The road file.
 * @param company The company to hold the roads in.
 */
void loadRoads(FILE * file, company *company);

#endif
```

4.7 company.c

```
/**
 * @brief The source file for a company.
 * @file company.h
 * @author Alexander Brown
 *
 * Created 18 November 2010
 *
 * Modified 09 December 2010 - Changed to use NULL instead of 0.
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "../lib/shortest.h"

#include "company.h"
#include "delivery.h"
#include "misc.h"
#include "region.h"
```

```
void loadCompany(FILE *file, company *company) {
  char *filename;
  FILE *datafile;

  company->name = calloc(MAX_NAME_SIZE, sizeof(char));
  fgets(company->name, MAX_NAME_SIZE-1, file);
  company->name = strtok(company->name, "\n");

  filename = calloc(MAX_NAME_SIZE, sizeof(char));
  fscanf(file, "%s", filename);
  datafile = fopen(filename, "r");
  if(NULL == datafile) {
    printf("Invalid region file: %s\n", filename);
    return;
  }
  else {
    loadRegion(datafile, &(company->region));
  }

  fscanf(file, "%s", filename);
  datafile = fopen(filename, "r");
  if(NULL == datafile) {
    printf("Invalid roads file: %s\n", filename);
    return;
  }
  else {
    loadRoads(datafile, company);
  }

  fscanf(file, "%s", filename);
  datafile = fopen(filename, "r");
  if(NULL == datafile) {
    printf("Invalid parks file %s\n", filename);
    return;
  }
  else {
    int i;
    company->numberOfParks = entryCount(datafile);
    company->parks = calloc(company->numberOfParks, sizeof(int));

    for(i=0;i<company->numberOfParks;i++)
    {
      fscanf(datafile, "%d", &(company->parks[i]));
    }
  }

  fscanf(file, "%s", filename);
  datafile = fopen(filename, "r");
  if(NULL == datafile) {
    printf("Invalid deliveries file %s\n", filename);
    return;
  }
  else {
    company->numberOfDeliveries = lineCount(datafile);
    company->deliveries = calloc(company->numberOfDeliveries, sizeof(delivery));
    loadDeliveries(datafile, company->deliveries, company->numberOfDeliveries);
  }

  fscanf(file, "%s", filename);
  datafile = fopen(filename, "r");
  if(0 == datafile) {
    printf("Invalid vans file %s\n", filename);
    return;
  }
  else {
    fscanf(datafile, "%f %f", &(company->runningCostEmpty), &(company->runningCostPerKg));
  }

  fclose(datafile);
  free(filename);
}

void loadRoads(FILE * file, company *company) {
  road *parent;
  road *cursor;
```

```
  company->roads = malloc(sizeof(road));

  cursor = company->roads;

  fscanf(file,"%d %d %d",&cursor->from,&cursor->to,&cursor->length);

  /* Seemingly infinite loop, but breaks when EOF is found */
  while(1) {
    parent = cursor;
    cursor = malloc(sizeof(road));
    if(fscanf(file,"%d %d %d", &cursor->from, &cursor->to, &cursor->length) != EOF){
      parent->next = cursor;
    }
    else {
      free(cursor);
      break;
    }
  }
}
```

4.8 region.h

```
/**
 * @brief The header file for regions.
 * @file
 * @author Alexander Brown
 *
 * Created 14 November 2010
 */

#include <stdio.h>

#ifndef MAX_NAME_SIZE
/**
 * The maximum size of any name string for a region or locations.
 */
#define MAX_NAME_SIZE 255
#endif

#ifndef _REGION
#define _REGION

/**
 * @brief The following structure defines a location.
 */
typedef struct
{
  /**
   * @brief The location code of the location defined by this structure.
   */
  int code;

  /**
   * @brief The name of this location.
   */
  char *name;
} location;

/**
 * @brief The following structure defines a region.
 */
typedef struct
{
  /**
   * @brief The name of this region
   */
  char *name;

  /**
   * @brief The number of locations held by this structure.
   */
  int numberOfLocations;

  /**
```

```
   * @brief The locations held by this structure.
   */
  location *locations;
} region;

/**
 * @brief Loads a region from a given file.
 * @param file The file to load from.
 * @param region The region to work with.
 */
void loadRegion(FILE * file, region * region);

/**
 * @brief Counts the number of locations in a location file (number of lines - 1).
 *
 * @param file The file to get number of lines of.
 *
 * @return The number of locations in a location file.
 */
int numberOfLocations(FILE * file);

#endif
```

4.9 region.c

```
/**
 * @brief The source file for regions and their operators.
 * @file
 * @author Alexander Brown
 *
 * Created 14 November 2010
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "misc.h"
#include "region.h"

void loadRegion(FILE * file, region *region) {
  int noLocations = numberOfLocations(file);
  int i;

  region->name = calloc(MAX_NAME_SIZE, sizeof(char));
  region->numberOfLocations = noLocations;
  fgets(region->name,MAX_NAME_SIZE, file);
  region->name = strtok(region->name,"\n");

  region->locations = calloc(noLocations,sizeof(location));

  for(i=0;i<noLocations;i++) {
    region->locations[i].name = calloc(MAX_NAME_SIZE, sizeof(char));
    fscanf(file,"%d ",&(region->locations[i].code));
    fgets(region->locations[i].name,MAX_NAME_SIZE-1,file);

    region->locations[i].name = strtok(region->locations[i].name,"\n");
  }
}


int numberOfLocations(FILE * file) {
  return lineCount(file)-1;
}
```

4.10 delivery.h

```
/**
 * @brief The header file for a delivery.
 * @file
 * @author Alexander Brown
 *
 * Created 18 November 2010
 */
```

```
#ifndef _DELIVERY_
#define _DELIVERY_

#include <stdio.h>

/**
 * @brief Defines a delivery.
 */
typedef struct {
  /**
   * The code for the retailer.
   */
  int retailer;

  /**
   * The location of the retailer.
   *
   * @see location.code
   */
  int location;

  /**
   * The number of deliveries per week.
   */
  int deliveries;

  /**
   * The weight of the delivery.
   */
  int weight;
} delivery;

/**
 * @brief Loads all deliveries from a file.
 * @param file The file to load from.
 * @param deliveries The deliveries to load to.
 */
void loadDeliveries( FILE * file , delivery *deliveries , int noDeliveries);

#endif
```

4.11 delivery.c

```
/**
 * @brief The source file for a delivery.
 * @file
 * @author Alexander Brown
 *
 * Created 18 November 2010
 */

#include <stdio.h>
#include <stdlib.h>

#include "delivery.h"
#include "misc.h"

void loadDeliveries( FILE *file, delivery *deliveries, int noDeliveries) {
  int i;
  rewind(file);

  for(i=0;i<noDeliveries;i++) {
    fscanf(file,"%d %d %d %d",
      &(deliveries[i].retailer),
      &(deliveries[i].location),
      &(deliveries[i].deliveries),
      &(deliveries[i].weight));
  }
}
```

4.12 misc.h

```
/**
```

```
 * @brief The header file for miscallanous functions.
 * @file
 * @author Alexander Brown
 *
 * Created 18 November 2010
 */

#include <stdio.h>

/**
 * @brief Counts the number of lines in a file.
 *
 * @param file The file to count lines of.
 */
int lineCount(FILE * file);

/**
 * @brief Counts the number of entries in a file.
 *
 * @note Counts the number of spaces in a file and adds one.
 *
 * @param file The file to count entries in.
 */
int entryCount(FILE * file);
```

4.13 misc.c

```
/**
 * @brief The source file for miscallanous functions.
 * @file
 * @author Alexander Brown
 *
 * Created 18 November 2010
 */

#include <stdio.h>
#include "misc.h"

int lineCount(FILE * file) {
  int lines = 0;
  int c;

  while((c=fgetc(file))!=EOF) {
    if(c=='\n')
      lines++;
  }

  rewind(file);
  return lines;
}

int entryCount(FILE *file) {
  int entries = 0;
  int c;

  while((c=fgetc(file))!=EOF)
    if(' ' == c)
      entries++;

  rewind(file);
  return entries+1; /* Add one for the last entry */
}
```