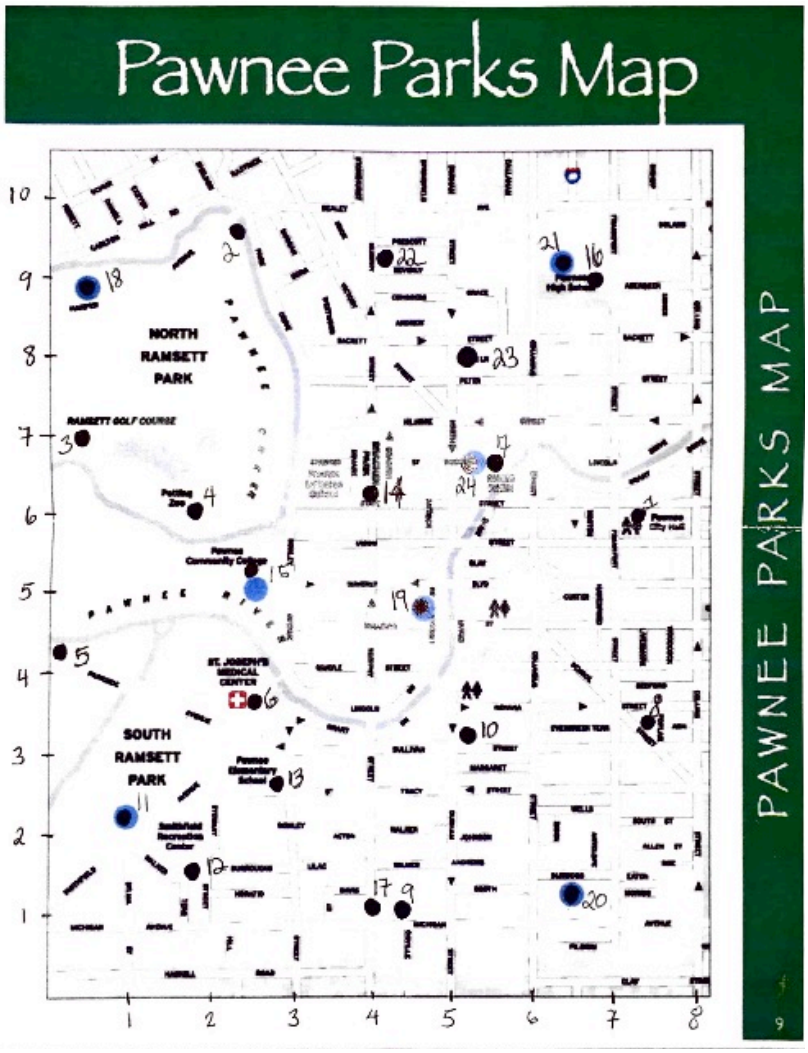
ECE 2036 Lab 3: "Making Pawnee a Better Place"

Due: Oct 5, 2018 @ 11:59PM
Reading: Deitel & Deitel Chapter 2-6

In this lab you will develop a basic software system to help government workers in Pawnee, Indiana be more efficient in their jobs. This effort is being spearheaded, of course, by Leslie Knope who has a very complex daily schedule. She has many places in Pawnee to visit every day as a part of her job as the assistant director for the Parks and Recreation Department. Her boss, Ron Swanson, is not very excited about this project, but has relented to Leslie's many demands to hire a Georgia Tech undergraduate student to help her optimize her travel time each day.

As seen below, Leslie has just faxed you a copy of a map of Pawnee and has roughly marked the 24 places in Pawnee that she might visit as a part of her job. The exact coordinates for each location appears in Appendix A.



The objectives of this lab are to give you practice:

1. Using basic C++ classes;
2. Creating basic arrays of user-defined objects;
3. Creating constructors and overloading constructors;
4. Using and creating set and get member functions in C++ classes;
5. Using C++ string objects;
6. Using basic text file I/O objects and operators; and
7. Exploring an optimization problem that can be difficult to solve (falls in the NP-complete category).

Software engineering design is about finding problems, having insight, and solving those problems. For this lab you might consider the following ideas to set the stage!

> **Problem:** Leslie Knope has many places in Pawnee that she needs to visit in one day, and she is always running out of time!!
>
> **Insight:** After observing her day, you and Anne realize that she seems to randomly pick places to go in Pawnee related to her to do list. She seems to waste a lot of time simply travelling haphazardly from one place to another (along with eating a lot of waffles at JJ's Diner).
>
> **Solution:** You decide that she needs a software system to optimize the order of the tasks that she needs to complete based upon their locations in Pawnee.

User Interface and System Requirements

The user interface is very basic, and it will prompt the user to enter the locations in Pawnee that he/she needs to visit in a given day. It is assumed that a Pawnee government employee, like Leslie Knope, will always start and end at City Hall. The user, however, does not have to enter City Hall as their first or last location because the software system will insert this automatically. Furthermore, you can assume that Leslie Knope will only put in one location at a time. Please note that the specific user input is indicated in **bold** below.

Please make your system prompts the user in the following way.

```
Good Morning Leslie Knope!!!

Please list the locations that you need to visit today
( input 0 to end your list )

1 (City Hall Start)
2
3
4
5
6
7
8
9
```

10
24
22
18
0

Input Requirements

In addition, to getting information from the user described above, your program will also need to read in the text file from Appendix A: PawneeMapCoordinates.txt. The X and Y coordinate of each location is needed in order to calculate the distance between locations. PLEASE calculate the distance between two locations using ORTHOGONAL distances only (i.e. the absolute value of the difference in the X coordinates added to the absolute value of the difference in the Y coordinates).

Output Requirements

Actually, I would like for you to output various paths that you calculate with different algorithms, and put the output data in a text file called `output.txt`. I would like for you to compare the paths produced by a simple heuristic algorithm, a greedy algorithm, and a simulated annealing algorithm. The output should be in the format in Appendix C, and I will describe each algorithm in Appendix D.

Programming Requirements

1. Please only use static (i.e. not dynamic) arrays in your classes. You can make them sufficiently large so that it can handle a full list of locations (i.e. 25). Later we will learn about dynamic arrays, but using them in classes at this point can be a little tricky!

2. I will require that you have at least the following classes listed below. I would like for you to have two source files and two corresponding header files for your classes. I will provide you a `main.cc` file that appears in Appendix B. Please make sure that your system works with the main file provided to you. You will need to have some inclusion guards in each header file. For example, in your `maps.h` you need to have the following additional preprocessor directives:

```
#ifndef MAPS_H
#define MAPS_H

//your class interfaces here

#endif
```

For your `schedule.h` you will need to have the following

```
#include <string>
#include "maps.h"
```

```
#ifndef SCHEDULE_H
#define SCHEDULE_H

//your class interface here

#endif
```

A. Classes in both maps.h (interface) and maps.cc (implementations)

MapLocation

This class will be called `MapLocation`. You will need to create a class with data members that hold the ID number for each location on the map, the name of the location, the x coordinate, and the y coordinate for the location. You can choose your own data member names. Please make sure that you have a full set of setters and getters for this class. Also, please have a default constructor and a constructor with 4 arguments.

Map

This class will be call `Map`. It will contain an array of `MapLocations` and the number of locations. For simplicity you can define a constant that has a fixed number of elements needed for this assignment. Actually, you will only need 24, but I am overallocated a little for this array. We have not talked about `const` designation for variables yet, so I will recommend another preprocessor directive that defines a constant that you will need to put in your header file.

```
#define NUM_LOCATIONS 25
```

Therefore, when you declare the array in this `Map` class, you can have something like the following:

```
MapLocation locations[NUM_LOCATIONS];
```

Later I will show you how to have a dynamic array to be managed inside a class. I will leave it up to you if you need setters or getters for this class.

void initializeMap(std::string)

This member function, which is called in `main.cc`, will need to read in the input file that has all the map coordinates and put the values in the `locations` array describe above.

B. Classes in both schedule.h (interface) and schedule.cc (implementations)

Schedule

This class will be called `Schedule`. You will need to create a class with data members that hold the employee's Name, number of locations to visit, and array of `MapLocations` for the list of locations that need to be visited, and a second array of `MapLocations` that has the `routeLocations` array that you create. If you would like to collapse this to just one array, then I am okay with this as well. You can choose your own data member names.

I would suggest that you have a set of "utility functions" that are private to Schedule that will be used internally in the class memberfunctions to generate the lists needed as specified in the output requirements of the program. For example, you might have something like the following in the class interface for the Schedule class.

```
private:
  float calcTotalDistance(MapLocation[]);
  void swap(MapLocation[], int, int);
  void createInitialRoute();
  void greedyOptimization();
  void SAOptimization();
  float calcDistance(MapLocation, MapLocation);
```

Furthermore, you will need to have public member functions to generate all the specified routes (i.e. generateRoute) and get the initial input from the user (i.e. getToDoList). Also, please have a default constructor and a constructor with a single string argument that sets the name of the person who the Schedule object belongs to.

Compiling Your Program

To compile and link all the files that I would like for you to manage. You will need to use the following command on coc-ice.pace.gatech.edu. (You can use a makefile as well if you so desire)

g++ maps.cc schedule.cc main.cc -o run_me_leslie_knope

APPENDIX A: PawneeMapCoordinates.txt

```
1  CityHall                7.2   5.9
2  NorthRamsettPark        2.5   9.6
3  RamsettGolfCourse       0.5   7.0
4  PettingZoo              1.8   6.1
5  PawneeRiverFront        0.2   4.3
6  StJosephMed             2.5   3.5
7  RiverPark                5.5   6.5
8  FoodandStuff            7.4   3.5
9  ThePit                  4.4   1.0
10 TheSnakeHole            5.1   3.1
11 SouthRamSettPark        1     2.1
12 SmithfieldRecCenter     1.9   1.5
13 PawneeElemSchool        2.9   2.6
14 BrakenPark              4     6.1
15 PawneeCommCollege       2.5   5.3
16 PawneeHighSchool        6.7   9
17 AnnesHouse              4.0   1
18 MouseRatConcert         0.5   8.8
19 FireDepartment          4.5   4.7
20 ToiletParty             6.5   1.2
21 JJDiners                6.2   9.1
22 PaunchBurger            4     9.1
23 TheBulge                5.1   7.7
24 TurnbillMansion         5.1   6.5
```

APPENDIX B: **main.cc**

```
#include "maps.h"
#include "schedule.h"

using namespace std;

int main()

{

//get input for coordinates and store in an internal array
//For simplicity you may assume that this is fixed at 24 locations

  Map PawneeMapCoordinates;
  PawneeMapCoordinates.initializeMap("PawneeMapCoordinates.txt");

//Now instantiate a LeslieKnopeSchedule object and get the locations
//that she needs to visit today. Remember she needs to
//begin and end at City Hall.

  Schedule LeslieKnopeSchedule("Leslie Knope");
  LeslieKnopeSchedule.getToDoList(PawneeMapCoordinates);
  LeslieKnopeSchedule.generateRoute();

}
```

APPENDIX C: example **output.txt** file

Todo list from worker

CityHall(1) to NorthRamsettPark(2) = 8.4

NorthRamsettPark(2) to RamsettGolfCourse(3) = 4.6

RamsettGolfCourse(3) to PettingZoo(4) = 2.2

PettingZoo(4) to PawneeRiverFront(5) = 3.4

PawneeRiverFront(5) to StJosephMed(6) = 3.1

StJosephMed(6) to RiverPark(7) = 6

RiverPark(7) to FoodandStuff(8) = 4.9

FoodandStuff(8) to ThePit(9) = 5.5

ThePit(9) to TheSnakeHole(10) = 2.8

TheSnakeHole(10) to TurnbillMansion(24) = 3.4

TurnbillMansion(24) to PaunchBurger(22) = 3.7

PaunchBurger(22) to MouseRatConcert(18) = 3.8

MouseRatConcert(18) to CityHall(1) = 9.6

TOTAL DISTANCE = 61.4

Create initial path with nearest neighbor hueristic

CityHall(1) to RiverPark(7) = 2.3

RiverPark(7) to TurnbillMansion(24) = 0.4

TurnbillMansion(24) to PettingZoo(4) = 3.7

PettingZoo(4) to RamsettGolfCourse(3) = 2.2

RamsettGolfCourse(3) to MouseRatConcert(18) = 1.8

MouseRatConcert(18) to NorthRamsettPark(2) = 2.8

NorthRamsettPark(2) to PaunchBurger(22) = 2

PaunchBurger(22) to StJosephMed(6) = 7.1

StJosephMed(6) to PawneeRiverFront(5) = 3.1

PawneeRiverFront(5) to TheSnakeHole(10) = 6.1

TheSnakeHole(10) to FoodandStuff(8) = 2.7

FoodandStuff(8) to ThePit(9) = 5.5

ThePit(9) to CityHall(1) = 7.7

TOTAL DISTANCE = 47.4

Greedy algorithm with random swapping

CityHall(1) to RiverPark(7) = 2.3

RiverPark(7) to TurnbillMansion(24) = 0.4

TurnbillMansion(24) to RamsettGolfCourse(3) = 5.1

RamsettGolfCourse(3) to MouseRatConcert(18) = 1.8

MouseRatConcert(18) to PaunchBurger(22) = 3.8

PaunchBurger(22) to NorthRamsettPark(2) = 2

NorthRamsettPark(2) to PettingZoo(4) = 4.2

PettingZoo(4) to PawneeRiverFront(5) = 3.4

PawneeRiverFront(5) to StJosephMed(6) = 3.1

StJosephMed(6) to ThePit(9) = 4.4

ThePit(9) to TheSnakeHole(10) = 2.8

TheSnakeHole(10) to FoodandStuff(8) = 2.7

FoodandStuff(8) to CityHall(1) = 2.6

TOTAL DISTANCE = 38.6

Simulated annealing (SA) algorithm with random swapping

CityHall(1) to RiverPark(7) = 2.3

RiverPark(7) to TurnbillMansion(24) = 0.4

TurnbillMansion(24) to PaunchBurger(22) = 3.7

PaunchBurger(22) to NorthRamsettPark(2) = 2

NorthRamsettPark(2) to MouseRatConcert(18) = 2.8

MouseRatConcert(18) to RamsettGolfCourse(3) = 1.8

RamsettGolfCourse(3) to PettingZoo(4) = 2.2

PettingZoo(4) to PawneeRiverFront(5) = 3.4

PawneeRiverFront(5) to StJosephMed(6) = 3.1

StJosephMed(6) to ThePit(9) = 4.4

ThePit(9) to TheSnakeHole(10) = 2.8

TheSnakeHole(10) to FoodandStuff(8) = 2.7

FoodandStuff(8) to CityHall(1) = 2.6

TOTAL DISTANCE = 34.2

APPENDIX D: Output Explanations

1. Todo List from Worker

This first section of your output uses the order from the user input as a path. You will need to make sure that you start at City Hall and end at City Hall on the list. You will output the orthogonal distance between each location, and then calculate and display the total distance travelled in one day. The units are kilometers. This output appears in APPENDIX C.

2. Create Initial Path with Nearest Neighbor Heuristic

For this section, I would like for you to start with City Hall and then find the NEXT location that is the closest. This will be the second place Leslie will travel. From that location, you will again try to find the closest location and go to that location. You will repeat this process in your code until all the locations have been visited. Finally, you can add City Hall to the end of your list.

This is not neccessarily an optimized list, but I would like this to be the starting configuration that you will use in the next two optimizations.

3. Greedy Algorithm with Random Swapping

For this optimization, I would like for you to use a "greedy" algorithm to see what effect it has on your optimization. In short, a greedy approach will only allow you to change the cost function such that you are only reducing the cost. The cost function in this case will be the total distance travelled to visit each of the locations just once while starting and ending at City Hall.

To do this, I would suggest that you randomly pick two locations (NOT CITY HALL LOCATIONS) and swap their positions in your `routeList`. After the swap, compare the total distance travelled to what it was BEFORE the swap. If the swap results in a REDUCTION in the cost function, then accept the swap. If the swap results in an INCREASE in the cost function, then you can consider it a "bad" swap and return the locations to their orginal position. A greedy algorithm reject all bad swaps! Repeat this for approximately 100 times the number of elements in your list.

Although this might sound like a reasonable approach to find the GLOBAL minimum in the cost function, it turns out that a greedy method that only accepts good swaps will typically get stuck in a LOCAL minimum. The next algorithm, which is called simulated annealing, is a powerful (but simple) technique to overcome this problem and allow you to find a solution that is closer to a global minimum.

4. Simulated Annealing (SA) Algorithm with Random Swapping

I will discuss this algorithm in class, but basically this algorithm is based upon observations in nature when producing cystalline material. A material that is in a cystal formation (i.e. perfect periodicity in all directions) is a material system that is in its lowest energy state. Energy in this analogy would be the "cost function" that we would like to minimize. To produce a material in

this state, experimentalists will heat up the material to a high temperature and then slowly cool the material according to what is called an "annealing schedule". This temperature schedule typically needs to be reduce slowly in order to reach a minimum energy state. If you cool it too quickly, the material will be amorphous and far from a crystalline state (and lowest energy).

The analogy with optimization is that you start the system at a high "temperature", which basically means that there is a high probability that you will accept "bad swaps." As you "cool" your optimization, the probability that you will accept "bad swaps" starts to decrease according to a Boltzmann probability distribution which looks something like the following

$$probability\ of\ energy\ change\ =\ e^{\frac{-(\Delta E)}{kT}}$$

This technique is actually quite easy to code. I will not worry about efficiency here, and I will provide you with some code that you can use. I have this as a private function of the Schedule class. You may have to adapt it slightly to make it work for your case. I assume that I have a RouteList array that is an array of MapLocations which is equal in size to the number of items that Leslie puts into her toDoList. You can play around with the annealing schedule to see if you can find a more optimized solution as well if you would like! ;-)

```
void Schedule::SAOptimization()
{
    float prob;
    float throwDice;
   //this is a linear ramping down of the temperature
   for (float temp = 15.0; temp >=0; temp-=0.1)
   {
   for (int i = 0; i < numItemsToDo*SCALE_FACTOR; i++)
   {
     //randomly pick two elements to swaps between
     // index 1 and numItemsToDo-1
     int pos1 = rand()%(numItemsToDo-1)+1;
     int pos2 = rand()%(numItemsToDo-1)+1;
     //I will allow pos1 and pos2 to be possibly equal for simplicity

     float dist1 = calcTotalDistance(RouteList);

     swap(RouteList, pos1, pos2);

     float dist2 = calcTotalDistance(RouteList);

     dist2 < dist1 ? prob =1.0 : prob = exp(-(dist2 - dist1)/(0.5*temp));

     throwDice = static_cast<float>(rand())/RAND_MAX;

     if (throwDice > prob) //reject the swap .. i.e. swap back
     {
       swap(RouteList,pos1,pos2);
     }
   }
  }//end temperature loop
}
```

APPENDIX E: Turnin Procedures

1. I would like for you to make a directory on coc-ice.pace.gatech.edu called `Lab3`. To make this directory, type the following while you are in your home directory.

   ```
   mkdir Lab3
   ```

2. Please go back to your home directory. You can do this by typing `cd` at the command prompt.

3. At the home directory, please type the following command to create a compressed `tarball` of your work

   ```
   tar -cvzf yourusernameLab3.tar.gz ./Lab3
   ```

4. As I showed you in class, you will need to download this file to your local machine. You will then submit this `yourusernameLab3.tar.gz` to canvas so that the TAs can grade it.

APPENDIX F: ECE 2036 Lab Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her lab; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

In addition, if a student's code does not compile, then he or she will have an automatic 30% deduction on the lab. Code that compiles but does not match the sample output can incur a deduction from 10% to 30% depending on how poorly the output matches the output specified by the lab. This is in addition to the other deductions listed below or due to the student not attempting the entire assignment.

AUTOMATIC GRADING POINT DEDUCTIONS

Element	Percentage Deduction	Details
Does Not Compile	30%	Program does not compile on pace ice cluster!
Does Not Match Output	10%-30%	The program compiles but doesn't match all output exactly

ADDITIONAL GRADING POINT DEDUCTIONS FOR RANDOMLY SELECTED PROGRAMS

Element	Percentage Deduction	Details
Correct file structure	10%	Does not use both .cc and .h files, implementing class prototype correctly
Encapsulation	10%	Does not use correct encapsulation in object-oriented objects
Setters/Getters	10%	Does not use setters and getters for each data member.
Constructors	10%	Does not implement constructors with the correct functionality.
Clear Self-Documenting Coding Styles	5%-15%	This can include incorrect indentation, using unclear variable names, unclear comments, or compiling with warnings. (See Appendix D)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count has one day; therefore H = 0.5*Hweekend

Appendix G: Good Programming Practices

Indentation
When using *if/for/while* statements, make sure you indent 2 to 4 spaces for the content inside those. For example…

```
for(int i; i < 10; i++)
  j = j + i;
```

If you have nested statements, you should use multiple indentions. Your *if/for/while* statement brackets *{ }* can follow two possible conventions. Each both getting their own line (like the *for* loop) OR the open bracket on the same line as the statement (like for the *if/else* statement) and closing bracket its own line. If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for(int i; i < 10; i++)
{
  if(i < 5) {
    counter++;
    k -= i;
  }
  else {
    k += i;
  }
  j += i;
}
```

Camel Case (Suggested But Not Required)
This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names
Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

Clear Comments
Some good opportunities to use comments are…

- Introducing a member function or class
- Introducing a section of code with long implementation
- Your name, class information, etc. at the beginning of the file

Appendix H: Input and Output Text Files

For this lab, you will do some basic manipulation of text files. In this section, I will show you how to instantiate an input or output text file object. In addition, you can use the insertion stream (<<) and extraction stream (>>) operators to send data to an output file or receive data from an input file, respectively.

Preprocessor Directive
You will need to have the following include statement in your header file that allows you to use the C++ standard library for I/O files.

```
#include <fstream>
```

Instantiating Output File Objects
To instantiate an output file object that you can use to manipulate your output text file, you will need something like the following.

```
std::ofstream YourOutputFileObject("outputfile.dat", std::ios::out);
```

The "outputfile.dat" name is an arbitrary identifier, and this constructor will create a file in the local directory that you execute your program in. The std::ios::out is a designation that you are creating an output file; any existing file with the same name will be overwritten.

Instantiating Input File Objects
To instantiate an input file object from which you can read data, you can do the following:

```
std::ifstream YourInputFileObject("inputfile.dat", std::ios::in);
```

Like the output file example, the "inputfile.dat" name is arbitrary and specifies the name of the file in the local directory from which you would like to read. As you see in the main file, you can use this file object with the ! operator to check to see if the file is valid.

For this lab, you will have the file in a C++ string. Therefore, you might have the beginning part of your initializeMap function look like:

```
void Map::initializeMap(string name)
{
  ifstream inputMap(name.c_str(), ios::in);

  //you will need more STUFF after this :-)
```

Insertion Stream Operator
Just like with the cout object, you can use the << operator to write data to an output file. You use the object name in place of cout in the following way.

```
YourOutputFileObject << "Hello File! " << std::endl;
```

Extraction Stream Operator
Furthermore, just like the cin object, you can use the >> operator to read data from an input file. You can use the object name in the following way.

```
YourInputFileObject >> string1;
```

This command will read a *single* string from the input file that is delineated by white space. For example, if the input file has the following text:

```
hello from professor Snape
```

The above line would have "hello" stored in `string1` with *no spaces*. Furthermore, the operation itself will return a true value if a string is successfully read in from the input file. This can enable you to embed this statement in a while loop condition to access all the strings in a file sequentially. For example, the following while loop will continue until all the strings have been read into the program one at a time.

```
while (YourInputFileObject >> string1)
{
   //manipulate value in string1
}
```

For this lab you will have to input various items from the input file provided to you in Appendix A. Therefore, you might have something like the following to pick out each piece of data.

```
while (inputMap >> tempID >> tempString >> tempX >> tempY)
{
  //you do something here
}
```