For this project, you will create a map object database like the one you did for Project 2. You will create a class to represent different objects on a map. Map objects on the map will be stored using a custom class you will write using a vector as the underlying data structure.

- 1. mapobject.cpp/hpp: Create a class called MapObject which has the following attributes:
 - label (string)
 - xloc & yloc (int)
 - speed (float)
 - direction (int)
 - type (object type t: an enum with CAR, TREE, POLICE, OBSTACLE, EMPTY)

and the following methods:

- an appropriate constructor. The default type should be EMPTY
- accessor/mutator methods for the attributes
- void get() prompts the user for information to fill a map object.
- void print() displays the record.
- collision(const MapObject &) returns true if both objects have the same location
- 2. mapobjectlist.cpp/hpp: Create a class called MapObjectList which stores an arbitrary number of map objects using one of the C++ STL container classes.
 - An accessor method (getter) that returns the vector where the map objects are stored:
 std::vector<MapObject> & getObjects() { return objects; }
 - void add(MapObject object) add a MapObject to the list
 - void print() prints a list of all objects
 - void get_all_of_type(object_type_t) prints the label, location, & direction of (all) the object(s) with the given type
 - void find(string) prints the location & speed of the object(s) with the given label
 - any other methods you see fit to implement
- 3. Define the following functions where you see fit. Note that they should not be a member of any class, but you can define them in any module that makes sense.
 - object_type_t string_to_type(std::string type_string) returns the object type that corresponds to the given string
 - char get_map_representation(object_type_t type) returns the character the given type should be displayd as. (EMPTY should return a space character).
- 4. Your code should work with the provided project2.cpp & map.cpp/hpp files. Print your name at the start and end of your programs output. I suggest using the provided makefile.

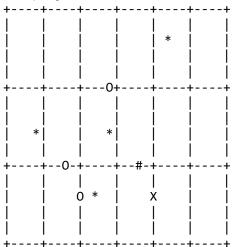
To receive full credit, your program should make good use of C++ object oriented design principles. Your MapObject and MapObjectList classes should be defined in separate .cpp/.hpp files. You should select appropriate types for all attributes & parameters. Your program should be written in C++, not C, so no printf, C-style strings, etc. Include a comment with your name at the top of each source file. Your code should compile using g++ with the flags used in class. Submit all source files required to build your program (including the input text file). Do **not** include any files in the zip file that are not required to compile your code (eg. .o, .exe, swapfiles, backups, etc.).

In addition to your zipped source, **upload a separate Word, .txt or PDF file** with a brief description of your implementation (including any issues you ran into and how you resolved them) along with the contents of your mapobject.h header file, your get() method from MapObject, and your find_all_of_type method().

Be prepared to answer questions demo your programs in class on the due date. Start early!

Sample Output:

```
$ ./project3.exe
```



print():

```
Type: CAR Label: RedCar Location: (10,12) Direction: 0 Speed: 65
Type: CAR Label: BlueCar Location: (14,5) Direction: 270 Speed: 45
Type: CAR Label: Bus Location: (8,10) Direction: 90 Speed: 55
```

Type: POLICE Label: Police1 Location: (18,10) Direction: 180 Speed: 55 Type: TREE Label: Tree1 Location: (22,2) Direction: 180 Speed: 0

Type: TREE Label: Tree1 Location: (22,2) Direction: 180 Speed: 0

Type: TREE Label: Tree2 Location: (4,8) Direction: 180 Speed: 0

Type: TREE Label: Tree3 Location: (12,12) Direction: 180 Speed: 0

Type: TREE Label: Tree4 Location: (14,8) Direction: 180 Speed: 0

Type: OBSTACLE Label: Mattress1 Location: (20,12) Direction: 180 Speed: 0

get all of type(CAR):

Type: CAR Label: RedCar Location: (10,12) Direction: 0 Speed: 65
Type: CAR Label: BlueCar Location: (14,5) Direction: 270 Speed: 45
Type: CAR Label: Bus Location: (8,10) Direction: 90 Speed: 55

find("RedCar")

Type: CAR Label: RedCar Location: (10,12) Direction: 0 Speed: 65

collision() test:
TEST1: SUCCESS!
TEST2: SUCCESS!