Homework 7
Assigned in Laboratory 11
Due Start of Laboratory 13

Please perform the following activities in groups of up to three people. Although you are allowed to talk with other people outside your group regarding assignment requirements and debugging, the work of each group must be its own—code must not be shared from one group to another.

Assignment files

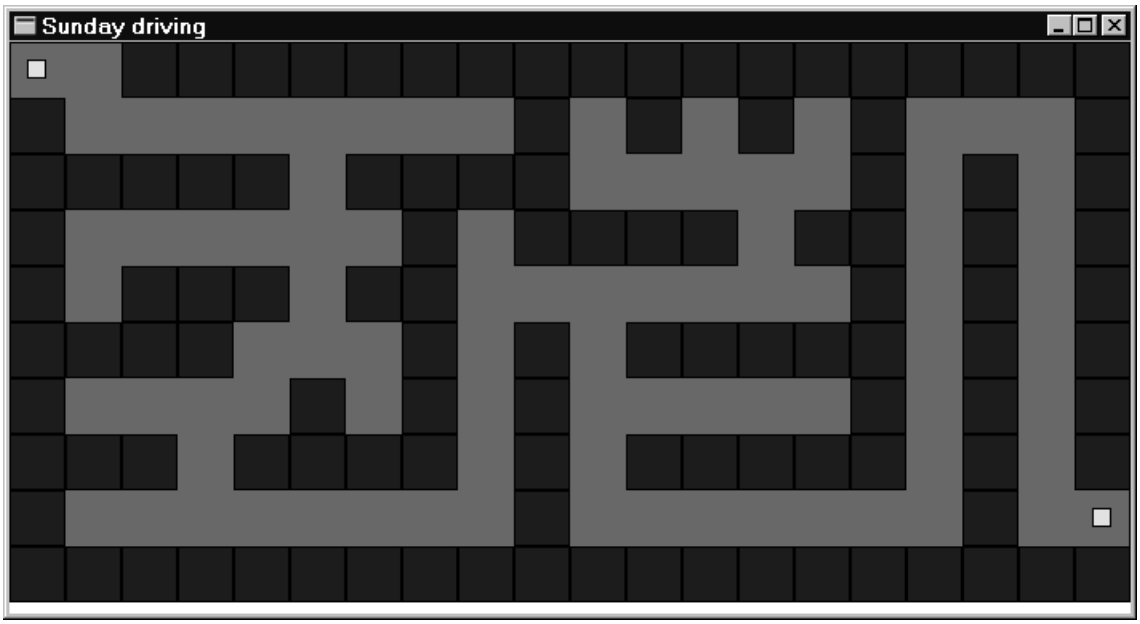
A self-extracting archive `hw07.exe` has been created for this assignment. Archive `hw07.exe` can be acquired by accessing the course home page `www.cs.virginia.edu/cs101` or on many of our university servers in the directory `F:\public\csfiles\cs101\hw`. If you are copying this file using a web browser, first copy the file to a hard drive and then use the file manager to copy the file to a floppy if need be.

Objective

Demonstrate array manipulation and member function implementation.

Problem statement

A popular gaming activity is traversing a maze to find a path from the starting point to finishing point. An example maze is shown in the following figure.

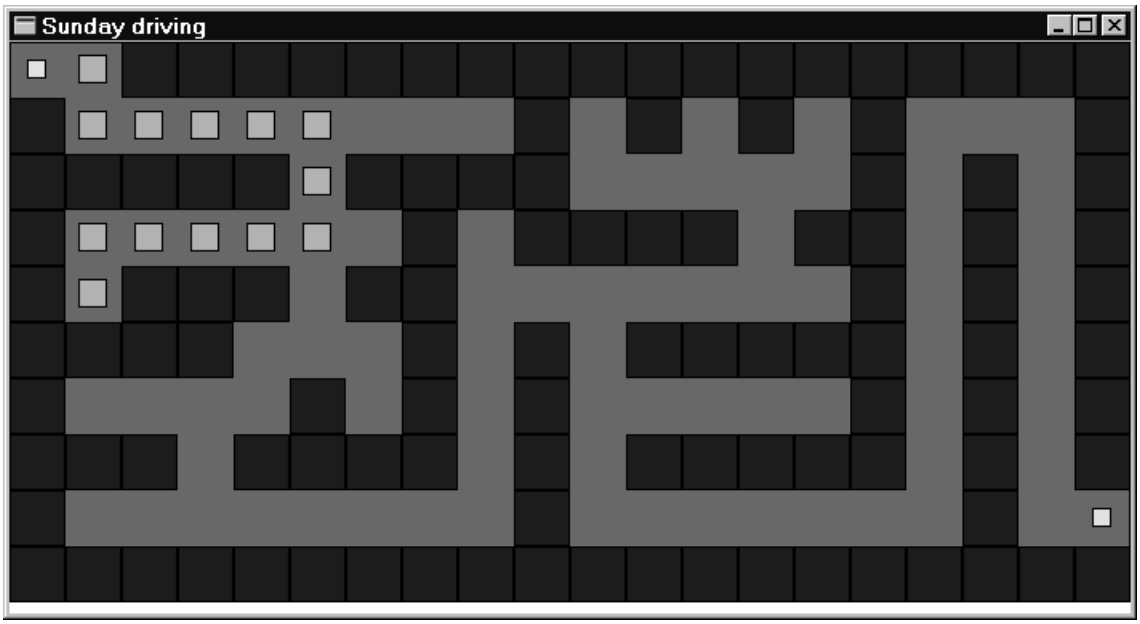


In this maze, the starting point is the small lightly colored spot in the upper left corner. The finishing point is the small square near the lower right corner. The dark areas represent *walls* and the lighter areas between the walls are *corridors*. The above maze has no free standing walls. A *free standing wall* is one that is not connected to the perimeter wall. A maze with no free standing walls is called a *proper maze*. The maze can be represented in part as a two-dimensional grid. Initially, each grid element is either an obstacle (part of a wall), free for traversing, starting point, or ending point. Later, some initially free grid elements will be marked seen.

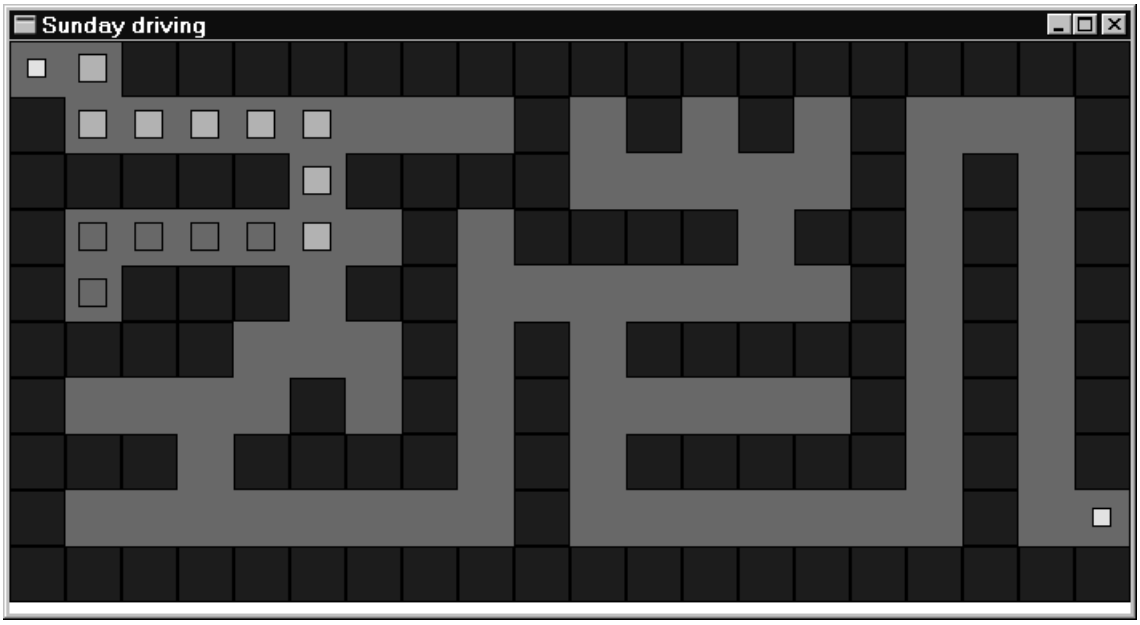
Proper mazes can be solved by using the right-hand strategy. The *right-hand strategy* says to walk along the maze with your right hand constantly sliding along the wall. By doing so, you will eventually reach the finishing point. Note that the right hand strategy does not guarantee that you will find the shortest path through the maze. This limitation does not matter for our purposes.

To have a notion of right-sidedness, we also need to have a notion of our current direction. We will consider north to be the direction that heads towards the top of the maze; south to be the direction that heads toward the bottom of the maze; west to be the direction that heads to the left side of the maze; and east to be the direction that heads to the right side of the maze.

Using the above maze and starting off facing east with the right-hand strategy, we will first step one-unit east, then step one-unit south, then step four-units east (taking individual steps), then step two-units south (taking individual steps), then step four-units west. We next step one-unit south. Our current progress is shown graphically in the following figure.

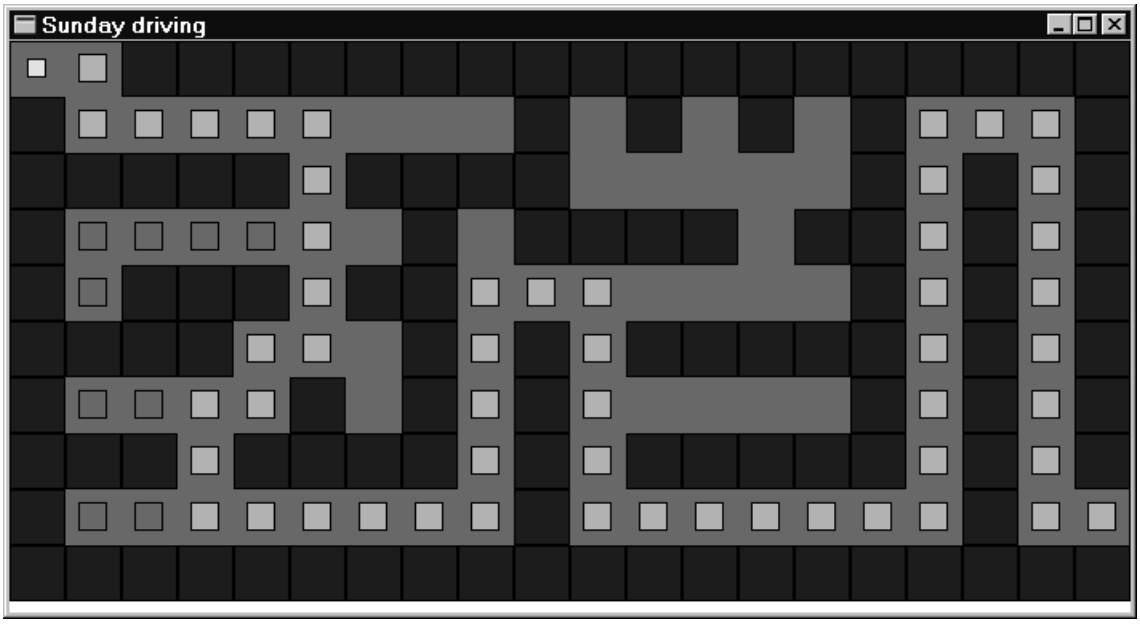


At this point the right-hand strategy forces us to retrace our steps in the opposite direction. However, when we reach the entrance of this dead end corridor, we will being facing east. (When we had first reached this point, we were facing south). Because we are facing east, our next step with the right-hand strategy will be one-unit south. The current situation is depicted in the following figure.



Observe in the preceding figure as we retrace our steps, only the outline of the errant steps is depicted.

A complete trace of where the right-hand strategy takes us is given in the following figure.



For this assignment, we will conduct a simulation of the right-hand strategy through a maze. For this simulation there are two important objects, the maze and the wanderer. We will develop classes for both such objects. A function `ApiMain()` will use these classes to instantiate a maze and a wanderer.

The simulation will be developed using the project file `drive.ide`. This project is composed of several files.

- `Maze.cpp`: Implementation of the `Maze` class. You will be required to complete several of the member functions.
- `Wander.cpp`: Implementation of the `Wanderer` class. You will be required to complete several of the member functions.
- `Drive.obj`: Contains a compiled-version of `ApiMain()` that controls the simulation. The compiled implementation of the right-hand strategy is in this file. Function `ApiMain()` will prompt the user for the name of a file that contains the description of the maze. A sample data file `test.dat` is supplied as part of the archive `hw07.exe`.
- `Ezwin.lib`: The compiled library version of the EzWindows library.

The files `maze.cpp` and `wander.cpp` include respectively the header files `maze.h` and `wander.h`. These header files along with the previously-mentioned `cpp` and `obj` files are part of the archive `hw07.exe`. The header files do not require any modification on your part. To get a sense of what you must implement, we provide listings of `maze.h` and `wander.h`.

File maze.h

```
enum LocationStatus { Free, Obstacle, Start, Finish, OutOfBounds, Seen};
class Maze {
   enum MaxMazeDimemsions { MaxRows = 20, MaxColumns = 30 };
   public:
      Maze(int length, int width, istream &sin, SimpleWindow &W);
      LocationStatus Status(int r, int c) const;
      void Display() const;
      int GetStartRow() const;
      int GetStartColumn() const;
      int GetFinishRow() const;
      int GetFinishColumn() const;
      void SetStatus(int r, int c, const LocationStatus &s);
   private:
      void DrawBackground() const;
      void DrawObstacle(int r, int c) const;
      void DrawStart() const;
      void DrawFinish() const;
      void SetAllFree();
      void ExtractStart(istream &sin);
      void ExtractFinish(istream &sin);
      void ExtractObstacles(istream &sin);
      LocationStatus Grid[MaxRows][MaxColumns];
      int NumberRows;
      int NumberColumns;
      int StartRow;
      int StartColumn;
      int FinishRow;
      int FinishColumn;
      SimpleWindow &MazeWindow;
};
```

File wander.h

```
enum Direction {North, South, East, West};
class Wanderer {
   public:
      Wanderer(int r, int c, SimpleWindow &W);
      void MoveNorth(Maze &M);
      void MoveSouth(Maze &M);
      void MoveEast(Maze &M);
      void MoveWest(Maze &M);
      void LookNorth(const Maze &M) const;
      void LookSouth(const Maze &M) const;
      void LookEast(const Maze &M) const;
      void LookWest(const Maze &M) const;
      Direction IsFacing() const;
      void Draw() const;
      int GetRow() const;
      int GetColumn() const;
   private:
      Direction Facing;
      int CurrRow;
      int CurrColumn;
      SimpleWindow &MazeWindow;
};
```

Library Maze implementation

The maze library defines the enumerated type `LocationStatus` to specify the following symbolic constants: `Free`, `Obstacle`, `Start`, `Finish`, and `OutOfBounds`. The library also defines the class `Maze`.

The class `Maze` has the following data members.

- `NumberRows`: Number of rows in the maze.
- `NumberColumns`: Number of columns in the maze.
- `StartRow`: Row coordinate of starting point.
- `StartColumn`: Column coordinate of starting point.
- `FinishRow`: Row coordinate of finishing point.
- `FinishColumn`: Column coordinate of finishing point.
- `MazeWindow`: Reference to the `SimpleWindow` in which all graphical displays are performed.
- `Grid`: A multidimensional array whose elements are of type `LocationStatus`. Element `Grid[r][c]` indicates the status of the `c`-th spot in the `r`-th row of the maze.

The class `Maze` has the following member functions already implemented.

- `Maze(int length, int width, istream &sin, SimpleWindow &W)`: This constructor uses length and width to set the number of rows in the maze; it uses input stream `sin` and member functions `ExtractStart()`, `ExtractFinish()`, and `ExtractObstacles()` to extract the makeup of the maze; it uses `W` to initialize `MazeWindow`.
- `Display()`: displays a graphical representation of the maze. It uses several private member functions that you must implement.
- `Status(int r, int c)`: Inspector that returns a `LocationStatus` value that shows how the `c`-th spot in the `r`-th row of the maze is being used. If the position does not lie on the maze, the value `OutOfBounds` is returned.
- `GetStartRow()`: Inspector that returns the row of the starting position.
- `GetStartColumn()`: Inspector that returns the column of the starting position.
- `GetFinishRow()`: Inspector that returns the row of the finishing position.
- `GetFinishColumn()`: Inspector that returns the column of the finishing position.

The class `Maze` needs the following member functions implemented by you.

- `DrawBackground()`: Draws a `Magenta` rectangle to `MazeWindow` whose dimensions match the size of the maze being represented. The center of this rectangle is the center of `MazeWindow`.
- `DrawObstacle(int r, int c)`: Draws a 1 by 1 `Blue` rectangle at location (`c` + 0.5, `r` + 0.5).
- `DrawStart()`: Draws a 0.35 by 0.35 `Yellow` rectangle at location (`c` + 0.5, `r` + 0.5) where `r` and `c` are the row and column locations of the starting point.
- `DrawFinish()`: Draws a 0.35 by 0.35 `Yellow` rectangle at location (`c` + 0.5, `r` + 0.5) where `r` and `c` are the row and column locations of the finishing point
- `SetAllFree()`: Sets all elements of `Grid` to `LocationStatus` `Free`.
- `ExtractStart(istream &sin)`: Extracts two integer values from the input stream `sin` and uses these values to set data members `StartRow` and `StartColumn`.
- `ExtractFinish(istream &sin)`: Extracts two integer values from the input stream `sin` and uses these values to set data members `FinishRow` and `FinishColumn`.
- `ExtractObstacles(istream &sin)`: Extracts until end of file is reached pairs of integer values. Each pair correspond to a row and column location of the maze whose `LocationStatus` value is `Obstacle`, i.e., the corresponding element of `Grid` is set to `Obstacle`.
- `SetStatus(int r, int c, const LocationStatus &s)`: Sets the `Grid` element corresponding to the `c`-th spot in the `r`-th row to status `s`.

Library Wanderer implementation

The wanderer library defines the enumerated type `Direction` to specify the following symbolic constants: `North`, `South`, `East`, and `West`. The library also defines the class `Wanderer` for representing a vehicle.

The class `Wanderer` has the following data members.

- `Facing`: Indicates which direction the vehicle is currently facing.
- `CurrRow`: Indicates the current row coordinate of the vehicle.
- `CurrColumn`: Indicates the current column coordinate of the vehicle.
- `MazeWindow`: Reference to the `SimpleWindow` in which all graphical displays are performed.

The class `Wanderer` has the following member functions already implemented.

- `GetRow()`: Inspector which returns the current row coordinate of the vehicle.
- `GetColumn()`: Inspector which returns the current column coordinate of the vehicle.
- `IsFacing()`: Inspector which returns the current direction of the vehicle.

The class `Wanderer` needs the following member functions implemented by you.

- `Wander(int r, int c, SimpleWindow &W)`: Initializes the vehicle so that its starting location is the `c`-th column of the `r`-th row. It does so by setting data members `CurrRow` and `CurrColumn`; it sets data member `Facing` to `East`. Reference data member `MazeWindow` to `W` in the member initialization list.
- `MoveNorth(Maze &M)`: Causes `CurrRow` to be updated to reflect the vehicle has moved northward one unit. Data member `Facing` is set to `North`. The status of the newly visited position in `Maze M` is sent to `Seen`. A move that would cause the vehicle to go off the maze causes an error message and no movement.
- `MoveSouth(Maze &M)`: Causes `CurrRow` to be updated to reflect the vehicle has moved southward one unit. Data member `Facing` is set to `South`. The status of the newly visited position in `Maze M` is sent to `Seen`. A move that would cause the vehicle to go off the maze causes an error message and no movement.
- `MoveEast(Maze &M)`: Causes `CurrColumn` to be updated to reflect the vehicle has moved eastward one unit. Data member `Facing` is set to `East`. The status of the newly visited position in `Maze M` is sent to `Seen`. A move that would cause the vehicle to go off the maze causes an error message and no movement.
- `MoveWest(Maze &M)`: Causes `CurrColumn` to be updated to reflect the vehicle has moved westward one unit. Data member `Facing` is set to `West`. The status of the newly visited position in `Maze M` is sent to `Seen`. A move that would cause the vehicle to go off the maze causes an error message and no movement.
- `LookNorth(const Maze &M)`: Returns the status of the maze element in `Maze M` to the immediate north of the vehicle's position.
- `LookSouth(const Maze &M)`: Returns the status of the maze element in `Maze M` to the immediate south of the vehicle's position.
- `LookEast(const Maze &M)`: Returns the status of the maze element in `Maze M` to the immediate east of the vehicle's position.
- `LookWest(const Maze &M)`: Returns the status of the maze element in `Maze M` to the immediate west of the vehicle's position.
- `Draw()`: A 0.5 by 0.5 `Cyan` rectangle is drawn to `MazeWindow` at location (`c` + 0.5, `r` + 0.5) where `r` and `c` are the current row and column locations of the vehicle.

Other notes

- Source file `wander.cpp` and `maze.cpp` are to be completed for your assignment. A hard copy of the two files are to be handed in at the start of the scheduled laboratory and an electronic version is to be submitted either prior to or at the start of that same lab. No other files should be submitted. If you have inadvertently submitted other files, please delete them. Do not delete the source files that you submitted for previous assignments. Your files should contain the standard header. It should properly identify all members of your group. It must contain the pledge. All group partners must be from the same lab section. There should be only one submission per group. The program should follow course programming style guidelines. The guidelines are contained in the course workbook and on the course web page.
- If you decide to develop your program on a non-ITC machine, you must make modifications to the project file. Instructions on how to modify a project file were given in the second programming assignment.