

Classes: A First Look

```
#include <iostream.h>
```

```
#define SIZE 10
```

```
// Declare a stack class for characters
```

```
class stack {
```

```
    char stck[SIZE]; // holds the stack
```

```
    int tos;          // index of top-of-stack
```

```
public:
```

```
    void init();          // initialize stack
```

```
    void push(char ch); // push character on stack
```

```
    char pop();          // pop character from stack
```

```
}
```

// Initialize the stack

```
void stack::init() { tos = 0; }
```

// Push a character.

```
void stack::push(char ch) {  
    if (tos==SIZE) { cout << "Stack if full"; return; }  
    stck[tos] = ch;  
    tos++; }
```

// Pop a character

```
char stack::pop() {  
    if (tos==0) { cout << "Stack is empty";  
                return 0; // return null on empty stack  
            }  
    tos--; return stck[tos]; }
```

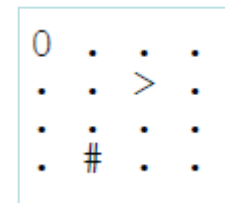
```
main() {  
    stack s1, s2; // create two stacks  
    int i;  
    // initialize the stacks  
    s1.init();  
    s2.init();  
  
    s1.push('a');      s2.push('x');  
    s1.push('b');      s2.push('y');  
    s1.push('c');      s2.push('z');  
  
    for (i=0; i<3; i++) cout << "Pop s1: " << s1.pop() << "\n";  
    for (i=0; i<3; i++) cout << "Pop s2: " << s2.pop() << "\n";  
  
    return 0;  
}
```

This page intentionally left blank



HW #4 Grid class (arrays)

- You will write a class called **Grid**, using **C++**, or **Java** or **Python** if you prefer.
- A Grid object will be made up of a grid of positions, numbered with rows and columns. Row and column numbering start at 0, at the top left corner of the grid.
- A grid object also has a "**mover**", which can move around to different locations on the grid. **Obstacles** (which block the mover) and **other objects** (that can be placed or picked up) are also available. Here is a sample picture of a Grid object in its display format:



0	.	.	.
.	.	>	.
.	#	.	.
.	.	.	.

A 4x4 grid display. The first cell (0,0) contains the number '0'. The cell at (1,2) contains the character '>'. The cell at (2,1) contains the character '#'. All other cells contain a period '.'.

0	.	.	.
.	.	>	.
.	.	.	.
.	#	.	.

HW #4 (2)

- This is a Grid object with 4 rows and 4 columns (numbered 0 – 3).
- The **mover** '>' is at row 1, column 2, facing east.
- The **obstacle** '#' is at row 3, column 1.
- The **other item** '0' is at row 0, column 0.
- The @ character indicates that the mover and an item (0) currently occupy the same position on the grid.
- **Grid.h** is defined as follows. Your member function prototypes are already given in this file. You will need to add appropriate member data. You will also need to define each of the member functions in the file **Grid.cpp**.

HW #4 (3)

- You may add whatever member data you need, but you must store the grid itself as a two-dimensional **array**. Maximum grid size will be 40 rows and 40 columns. (Note that this means dynamic allocation will NOT be needed for this assignment).

// Grid class

```
class Grid {  
public: // public static class constants, for direction indicators  
    static const int NORTH = 0;  
    static const int WEST = 1;  
    static const int SOUTH = 2;  
    static const int EAST = 3;
```

HW #4 (4)

// public member functions

Grid(); // build 1 x 1 grid with mover in only square, facing east

Grid(int r, int c); // build grid with r rows, c cols,
// blocks around edge with random exit
// and random mover position and direction

Grid(int r, int c, int mr, int mc, int d);
// build empty grid with r rows, c cols, and
// mover at row mr, col mc, and facing direction d

bool Move(int s); // move forward s spaces, if possible

void TogglePath(); // toggle whether or not moved path is shown

void TurnLeft(); // turn the mover to the left

HW #4 (5)

```
void PutDown();           // put down an item at the mover's position
bool PutDown(int r, int c); // put down an item at (r, c), if possible
bool PickUp();            // pick up item at current position
bool PlaceBlock(int r, int c); // put a block at (r, c), if possible
void Grow(int gr, int gc); // grow the grid by gr rows, gc columns
```

```
void Display() const; // display the grid on screen
```

```
// Accessors
```

```
bool FrontIsClear() const; // check to see if space in front of mover is clear
bool RightIsClear() const; // check to see if space to right of mover is clear
int GetRow() const;        // return row of the mover
int GetCol() const;        // return column of the mover
```

HW #4 (6)

```
int GetNumRows() const; // return number of rows in the grid
int GetNumCols() const; // return number of columns in the grid

private:
    .....
};
```

- Meaning of Grid symbols
 - empty spot on the grid
 - 0** an item that can be placed, or picked up
 - #** a block (an obstacle). Mover cannot move through it

HW #4 (7)

- < mover facing WEST
 - > mover facing EAST
 - ^ mover facing NORTH
 - v mover facing SOUTH
 - @ mover occupying same place as item (0)
-
- A printed space ' ' in a grid position represents a spot that the mover has already moved through when the path display is toggled ON.

HW #4 (8)

1. **Grid()** : The default constructor should create a 1 x 1 grid, with the mover in the only position, facing EAST.
2. **Grid(int r, int c)** : The two-parameter constructor will accept two integers, representing rows and columns. Create a grid with r rows and c columns. If either of these is less than 3, default it to 3. If either is greater than the max number of rows or columns, default it to the max number. This grid should be built such that blocks are placed all around the edge, with one random exit (i.e., with no block).

HW #4 (9)

- The mover should be in a random position and facing a random direction within the grid. When setting up the randomness, make sure each possibility has an equal chance of happening. For example, the random direction has 4 possibilities, so each one should happen about 25% of the time. For the random exit, it will be sufficient to pick a random wall first, then pick a random location on that wall (note that the exit **cannot** be in a corner spot).

HW #4 (10)

- You will need the library **cstdlib** for the **srand** and **rand** functions. While it is not normally the best place to do it, you can go ahead and seed the random number generator here in the constructor in some appropriate way so that it is different for separate program runs.
- 3. Grid(int r, int c, int mr, int mc, int d) :** This constructor (5 parameters) takes in the following information, in this order:
- **r** : number of starting rows for the grid (if out of range, adjust like in the 2-parameter constructor, although minimum in this case is 1).

HW #4 (11)

- **c** : number of starting columns for the grid (if out of range, adjust like in the 2-parameter constructor, although minimum in this case is 1).
- **mr** : The starting row position of the mover (if out of range, adjust to the first or last row, whichever is closer).
- **mc** : The starting column position of the mover (if out of range, adjust to the first or last column, whichever is closer).
- **d** : The starting direction of the mover.
- Build the starting grid based on the incoming parameters and their descriptions above. Other than the mover, this grid starts out **empty**.

HW #4 (12)

- 4. Display()** : This function should print out "**The Grid:**" on one line, then output the full grid below it – place spaces between columns so that outputs line up more evenly. End with a newline (so that any next output will be on a separate line). If the path setting is toggled to ON, then any position the mover has already moved through should show up blank. If the path setting is toggled to OFF, then all empty grid spaces show up as dots '.'.
- Examples of the full Display format can be seen in the sample run for **test.cpp**.

HW #4 (13)

- For description of the path setting, see the function TogglePath() below.

5. TogglePath() : This function, when called, should reverse whatever the current "path" setting is. The path setting can be either ON or OFF. If it is ON, it means that displays of the grid should indicate where the mover has been by showing those positions as spaces. If the path is OFF, then all blank spots on the grid show as the dot character '.' no matter what. The initial default setting for any grid should be ON.

HW #4 (14)

6. Simple Accessors (Getters) : These are "getter" functions that should return the requested information:

- GetRow() should return the current row of the mover.
- GetCol() should return the current column of the mover.
- GetNumRows() should return the number of rows in the grid.
- GetNumCols() should return the number of columns in the grid.

HW #4 (15)

7. Predicate functions : These two functions return boolean values (true or false) to answer simple questions:

- `FrontIsClear()` should return an indication of whether the space in front of the mover is clear (i.e., not blocked and on the grid).
- `RightIsClear()` should return an indication of whether the space to the right of the mover is clear (i.e., not blocked and on the grid).

HW #4 (16)

8. Placing blocks and / or items : These functions involve placing things on the grid:

- **PutDown():** This function should place an "item" at the current location of the mover. If the location already contains an item, then nothing changes (i.e., the item is still there).
- **PutDown(int r, int c):** This function should place an "item" at position (r, c) where r is the row and c is the column. This function should return true for a successful placement, and false for failure.

HW #4 (17)

- For successful placement, the position has to exist within the grid boundaries and not already contain a block or another item. (It can, however, be placed in a spot where only the mover is located).
 - **PlaceBlock(int r, int c)**: This function should place a "block" at position (r, c) where r is the row and c is the column. This function should return true for a successful placement, and false for failure. For successful placement, the position has to exist within the grid boundaries and be an empty space (i.e., not containing a block, an item, or the mover).

HW #4 (18)

- 9. PickUp()** : This function should pick up the "item" at the mover's position. This means removing it from the grid. This function should return true for a successful pick-up (i.e., the item is at the mover's current position), and false for failure (there is no item there).
- 10. Move(int s)** : This function should move the mover forward, in the current direction, for s spaces. Return true for success, false for failure. A successful move must be a positive number of spaces, must remain on the grid, and must not attempt to move through a "block". On a failed move, the mover should remain in the original position. (i.e., the move must be all or nothing, based on the number of spaces given by the parameter).

HW #4 (19)

11. TurnLeft() : This function will cause the mover to turn 90 degrees to the **left**, from whichever way it is currently facing. (example: if the mover is facing NORTH, then TurnLeft() will make it face WEST).

12. void Grow(int gr, int gc) : This function should increase the size of the grid by gr rows and gc columns. If the increase causes either the number of rows or columns to exceed the maximum, simply set that (rows or columns) to the maximum. The grow should not change the current position of any item, mover, or block on the grid.

HW #4 (20)

- Note that the list of member functions given in the provided starter file (and described above) constitute the entire public interface of the class. These should be the **ONLY** public member functions in your class. If you create any extra member functions as helpers, put them in the **private** section of the class.
- You are being provided one starter test program, along with the sample output. You can use this to help test your class, but keep in mind that this is NOT a comprehensive set of tests.

// **test.cpp**: A small program that builds a simple maze in a grid, then has the mover navigate through the maze to pick up an item. It also drops a couple of other items on the way.

```
#include <iostream>
```

```
using namespace std;
```

```
#include "Grid.h"
```

```
int main() {
```

```
    // set up the initial grid
```

```
    Grid g(9,15,2,13,Grid::WEST);
```

```
    g.PutDown(2,1);
```

```
    for (int i = 0; i < 7; i++) {
```

```
        g.PlaceBlock(i,11); g.PlaceBlock(i,7); g.PlaceBlock(i,3);
```

```
        g.PlaceBlock(i+2,5); g.PlaceBlock(i+2,9);
```

```
    }
```

```
    g.Display();
```

A

// now start moving

B g.TurnLeft(); g.Move(5); g.Display();

C g.TurnLeft(); g.TurnLeft(); g.TurnLeft(); g.Move(1); g.Display();

for (int i = 0; i < 2; i++) {

D g.Move(2); g.Display();

I

E g.TurnLeft(); g.TurnLeft(); g.TurnLeft(); g.Move(6); g.Display();

J

F g.PutDown(); g.Display();

K

G g.TurnLeft(); g.Move(2); g.Display();

L

H g.TurnLeft(); g.Move(6); g.Display();

M

g.TurnLeft(); g.TurnLeft(); g.TurnLeft();

}

N g.Move(3); g.Display();

O g.TurnLeft(); g.TurnLeft(); g.TurnLeft(); g.Move(5); g.Display();

P g.PickUp(); g.Display();

Q g.TurnLeft(); g.Move(1); g.Display();

R g.TogglePath(); g.Display();

S g.TogglePath(); g.Display();
}

Output.txt (1)

A

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # . . . # . . .
. 0 . # . # . # . # . # . < .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . . . # . . . # . . . . .
. . . . . # . . . # . . . . .
```

B

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # . . . # . . .
. 0 . # . # . # . # . # . . .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . . . # . . . # . . . v .
. . . . . # . . . # . . . . .
```

C

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # . . . # . . .
. 0 . # . # . # . # . # . . .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . . . # . . . # . . . < .
. . . . . # . . . # . . . . .
```

D

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # . . . # . . .
. 0 . # . # . # . # . # . . .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . . . # . . . # . . . < .
. . . . . # . . . # . . . . .
```

E

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # . . . # . . .
. 0 . # . # . # . # . # . . .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . . . # . . . # . . . ^ .
. . . . . # . . . # . . . . .
```

F

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # . . . # . . .
. 0 . # . # . # . # . # . . .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . # . # . # . # . # . # .
. . . . . # . . . # . . . @ .
. . . . . # . . . # . . . . .
```

Output.txt (2)

G

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # < 0 # . . .
. 0 . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . . . # . . . # . . .
. . . . . # . . . # . . .
```

H

```
The Grid:
. . . # . . . # . . . # . . .
. . . # . . . # . . . # . . .
. 0 . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . . . # . . . v # . . .
. . . . . # . . . # . . .
```

I

```
The Grid:
. . . # . . . . # . . . # . . .
. . . # . . . # . . . 0 # . . .
. 0 . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . . . # < # . . . # . . .
. . . . . # . . . # . . .
```

J

```
The Grid:
. . . # . . . ^ # . . . # . . .
. . . # . . . # . . . 0 # . . .
. 0 . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . . . # . . . # . . .
```

K

```
The Grid:
. . . # . . . @ # . . . # . . .
. . . # . . . # . . . 0 # . . .
. 0 . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . # . # . # . # . . .
. . . . . # . . . # . . .
```

Output.txt (3)

L

```
The Grid:
. . # . . # . . . # . . .
. . # < 0 # . 0 # . . .
. 0 . # . # # # . . .
. . . # . # # # . . .
. . . # . # # # . . .
. . . # . # # # . . .
. . . # . # # # . . .
. . . . # # # . . .
. . . . # . . . . .
```

M

```
The Grid:
. . # . . # . . . # . . .
. . # 0 # . 0 # . . .
. 0 . # # # # . . .
. . . # # # # # . . .
. . . # # # # # . . .
. . . # # # # # . . .
. . . # # # # # . . .
. . . . v # # . . .
. . . . # . . . . .
```

N

```
The Grid:
. . # . . # . . . # . . .
. . # 0 # . 0 # . . .
. 0 . # # # # . . .
. . . # # # # # . . .
. . . # # # # # . . .
. . . # # # # # . . .
. . . # # # # # . . .
. < . # # # . . .
. . . . # . . . . .
```

```
The Grid:
. . # . . # . . . # . . .
. . # 0 # . 0 # . . .
. @ . # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . . # . . . # . . .
```

O

```
The Grid:
. . # . . # . . . # . . .
. . # 0 # . 0 # . . .
. ^ . # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . # # # # # # . . .
. . . # . . . # . . .
```

P

Output.txt (4)

Q

The Grid:

```
. . . # . . . # . . . # . . .  
. . . # . . 0 # . . 0 # . . .  
< . . # . # # # # . . .  
. . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . . . # . . . . .
```

R

The Grid:

```
. . . # . . . # . . . # . . .  
. . . # . . 0 # . . 0 # . . .  
< . . # . # . # . # . # . . .  
. . . # . # . # . # . # . . .  
. . . # . # . # . # . # . . .  
. . . # . # . # . # . # . . .  
. . . # . # . # . # . # . . .  
. . . # . . . # . . . . .
```

S

The Grid:

```
. . . # . . . # . . . # . . .  
. . . # . . 0 # . . 0 # . . .  
< . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . # # # # # . . .  
. . . # . . . # . . . . .
```