TABLE OF CONTENTS

User Manual

Project 1: setting_the_table

This program will allow the user to enter letter (a, b, c, d or e) art via command line using a series of menu commands to control a "combination lock". The users' enter has ability, which is capable of multiple types (ASCII characters i.e. a b c d e or A B C D E), and a state variable that keeps track of whether or not the person is entering right password. On the other hand, purpose of finding right key is to have print out LOCK, UNLOCK and sound the alarm along the way in order to track down the right keys enters. If the users enters is right as in the give format of password, it will print UNLOCK on the screen. The enters should be only one character each time when it appear prompt. If the user inter wrong key, screen will sound and prompt user to keep entering. In addition, the dimension of maze compound can be changeable by users when the standard size is 10 X 10 as binary codes will be provided in txt file which build the size of the tables.

Executing the Program

Turn on and boot your computer to the operating system.

Download and locate the file CS232_P1.exe from your source. Double click the P3.exe file to run it. Alternatively, load the file from a DOS prompt by entering the file path along with the filename (P1) and extension (.exe).

Example:

C:\P1.exe

A console window will appear (or the program will execute in your console if run from DOS) and ask you for input.

Input

The program will display to the user a list of user dimension according to their input by default, as well as displaying a menu. The user will be presented with correct characters as password options in the menu and will be asked to input the characters value associated with the menu command. The default menu options are as follow:

Input the character

The user will be prompted to enter repeated characters till correct keys. After this menu is displayed it will be displayed again unless the user restart program or entered the correct keys when prompted for input. The user may enter any alphabets of lower case or upper case of commands as they wish in order to produce a drawing suited to their footprint. The commands may be entered one at a time or all at once separated by spaces. For example, if the user enter wrong password, the program will display to keep Wrong password and keep asking till the users enter the right keys.

Output

The output of the program will vary depending on the inputs the user gives. For example message will display as in the image displayed below.

Programmer Manual

Project 1: setting_the_table

Problem Description

This program implements a combination lock program, which uses ASCII characters as keys and a command line console as the medium. The program uses multiple custom ADTs to accomplish the task of implementing the concept of entering in a program as it right keys when users enter wrong keys the program will keep prompting for users to enter the right keys. As the users enter the right key, it will display "UNLOCK" ,otherwise; the message will be displayed as WRONG PASSWORD on the screen. Depending on whether or not the users enter the keys the message will be displayed. More information about the objects used may be found in their respective manuals.

Data Types

The data types used in this program are a combination of basic C and C++ data types as well as 5 class objects. The 5 classes all function independently of each other, allowing for modularity of objects. In this program, however, they are used together to create the turtle graphics drawing program as described above.

Class Objects:

class print

class Table

class insert

class remove

class lookUp

class map

Programmer Manual

2.1 : Class Table

Problem Description

This class is an Abstract Data Type (ADT), which represents a floor or canvas. It is implemented as a dynamic character array with mutator and accessor functions that allow for resizing of the dimension, as well as retrieving and changing individual elements (tiles) of the array.

Data Types

There are only two data types in the Floor class besides scratch variables. These are of type int and char*.

2.1Variables

char *maze_ - A pointer to an array of characters which is used to represent the floor. Each individual character can be thought of as a dimension in compound.

int privateWidth - The width of the compound, stored as an integer

int privateHeight - The height of the compound, stored as an integer

Member Functions

Besides the default and non-default constructors, copy constructor, and destructor, the Floor class has the following member functions.

Public

```
Table ();                                   //Default constructor
Table(int width, int height);  //Nondefault constructor
Table(int width, int height, char wallChar, char fillChar, char pathChar,
char startChar, char exitChar);
Table(const Maze &init);                //Copy constructor
~Table();                                   //Deconstructor

void fillTable();           //Fills the maze with default walls/fill
void printTable();                  //Prints the maze to console
void setStartPoint(int column, int row);    //Sets a start point
void setExitPoint(int column, int row);             //Sets an exit point
bool getFileFromFile(string filename);              //Creates a maze from txt
file
bool saveToFile(string filename);           //Saves a maze to txt file
char getWallType(int x, int y);                                 //Returns the
proper ASCII character for

//the wall at the given position
bool findAnExit(bool showPath, int delay);      //Algorithm to find the
exit
```

void setDimensions(int width, int height); //Resizes the maze
void setTable(int column, int row); //Creates a wall at the given position
bool isEnd(); //Checks to see if

bool isFree(); //Checks to see if the mouse is free
void pushSurrounds(); //Pushes the surrounds of the mouse
void moveMouse();

Private

void fillRWr(char *floor, int width, int height)

Fills a character array passed in as an argument with spaces.

Programmer Manual

2.2: Class insert

Problem Description

This class is an Abstract Data Type (ADT), which represents a table direction to insert keys. It is implemented as two variables to keep track of the state and brush type, with mutator and accessor functions that allow for changing the state and character.

Data Types

There are only two data types in the Pen class besides scratch variables. These are of type bool and char.

2.1Variables

char keys - A variable to hold an ASCII character that can be used as a brush in a character array

bool FSState_ - The up/down state of the table

Member Functions

Besides the default constructor and destructor, the Pen class has the following member functions.

3.1Public

void setState(bool isUp) variable	- Mutator for upState_
bool isUp() variable	- Accessor for the upState_
void set(char)	- Mutator for brush_
char getBrush()	- Accessor for brush_
void FSMState() upState_	- Mutator that toggles the

Programmer Manual

2.3: Class remove

Problem Description

This class is an Abstract Data Type (ADT), which represents a direction; Up, Down, Left, or Right to remove. It is implemented as a facing integer and an enumeration to correlate the directions with integer. It has mutator and accessor functions that allow for retrieval or changing of the direction by passing in string literals or string variables, as well as

retrieving an ASCII arrow character that will point in the direction of the facing.

Data Types

There is only one data type in the Direction class besides scratch variables. These are of type int.

2.1Variables

int keyDirection - The integer representation of the direction. The enumeration for the facing is as follows. NORTH = 0, EAST = 1, SOUTH = 2, WEST = 3

Member Functions

Besides the default constructor and destructor, the Direction class has the following member functions.

3.1Public

void setFacing(char* facing) - Mutator for facing_

char* getFacing() - Accessor for the facing as a string

char getAscii() - Accessor for the facing as an ASCII arrow

void () - Changes the facing to rotate clockwise

Programmer Manual

2.5: Class lookUp

Problem Description

This class is an Abstract Data Type (ADT), which represents a position on a Cartesian plane. It is implemented as two integers to keep track of x and y values that can be manipulated through mutator and accessor functions.

Data Types

There is only one data type in the Position class besides scratch variables. These are of type int.

2.1Variables

int index - mapping function as key

const key aKey - The position

Member Functions

Besides the default constructor and destructor, the Position class has the following member functions.

3.1Public

void setX(int x) - Mutator for x

int getX() - Accessor for x

void setY(int y) - Mutator for y

int getY() - Accessor for y

void setPosition(int x, int y) - Mutator for x and y at once

Programmer Manual

2.6: Class map

Problem Description

This class is an Abstract Data Type (ADT), which represents combination of lock with characters value. It is implemented using other objects created in this project such as Direction, Position, and Pen. The

turtle is designed to work with the Floor class but is not dependent on it and may be used with a simple character array.

Data Types

There is one basic C data type in the Turtle class, which is type char. It also contains three Abstract Data Types, transition, action, and Position.

2.1Variables

2.1.1 Private

char image_ - A variable to store the character used to represent the action

char imageUnder_ - A variable to store the character that the key is currently replacing in a character array or Floor object.

2.1.2 Public

Position position - Position object that keeps track of the keys position information

tansition Keys - Pen object that keeps track of information about the state of the keys

action direction - Direction object that keeps track of information about the turtles current facing/direction

Member Functions

Besides the default constructor and destructor, the Turtle class has the following member functions.

3.1Public

char getImage() - Accessor for the key image_

void setImage(char image) - Mutator for the key image_

char getImageUnder() - Accessor for the the key imageUnder_ variable

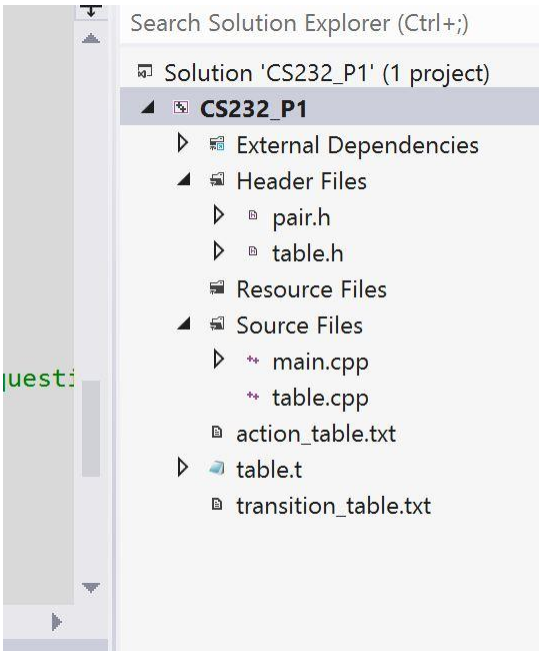
char setImageUnder(char imageUnder) - Mutator for the turtle's imageUnder_ variable

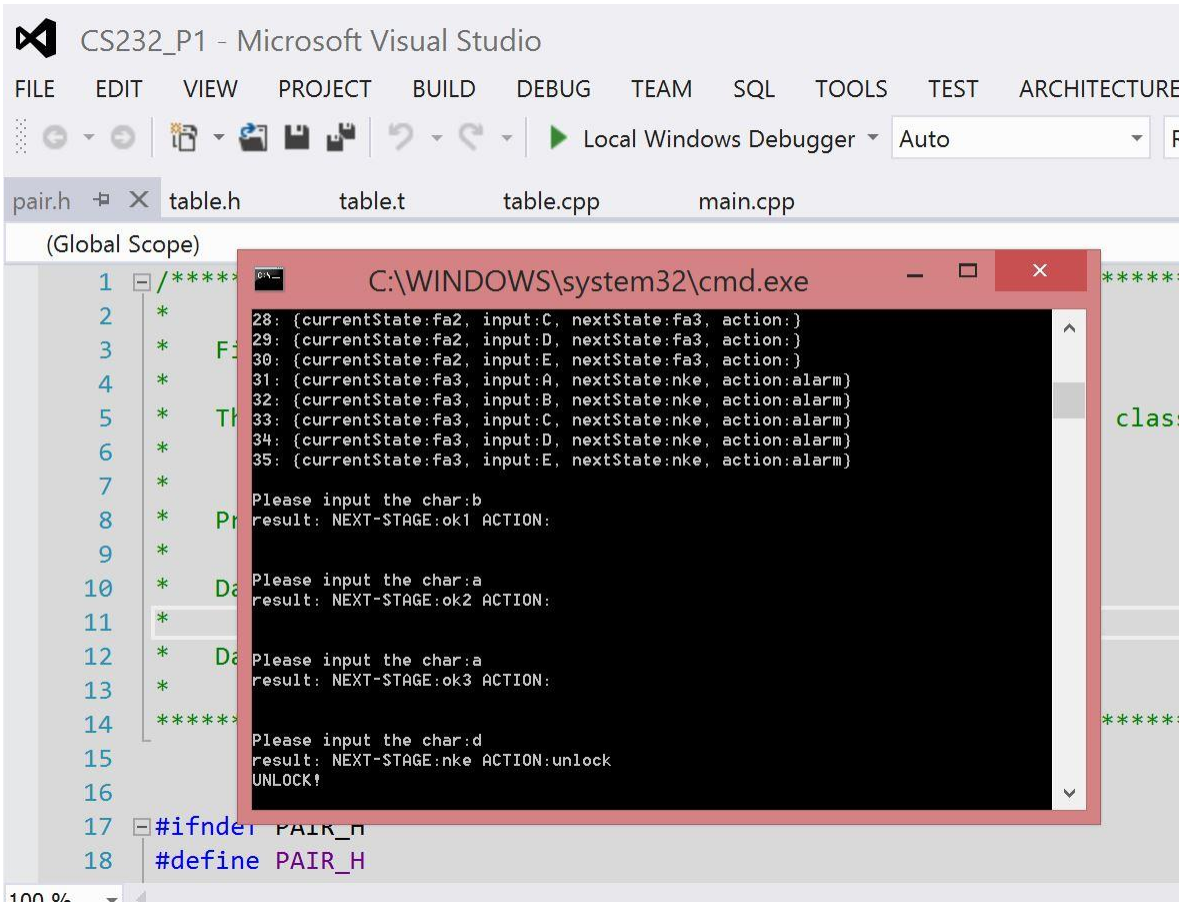
void move() - Function that increments the key Position based on the transition direction.

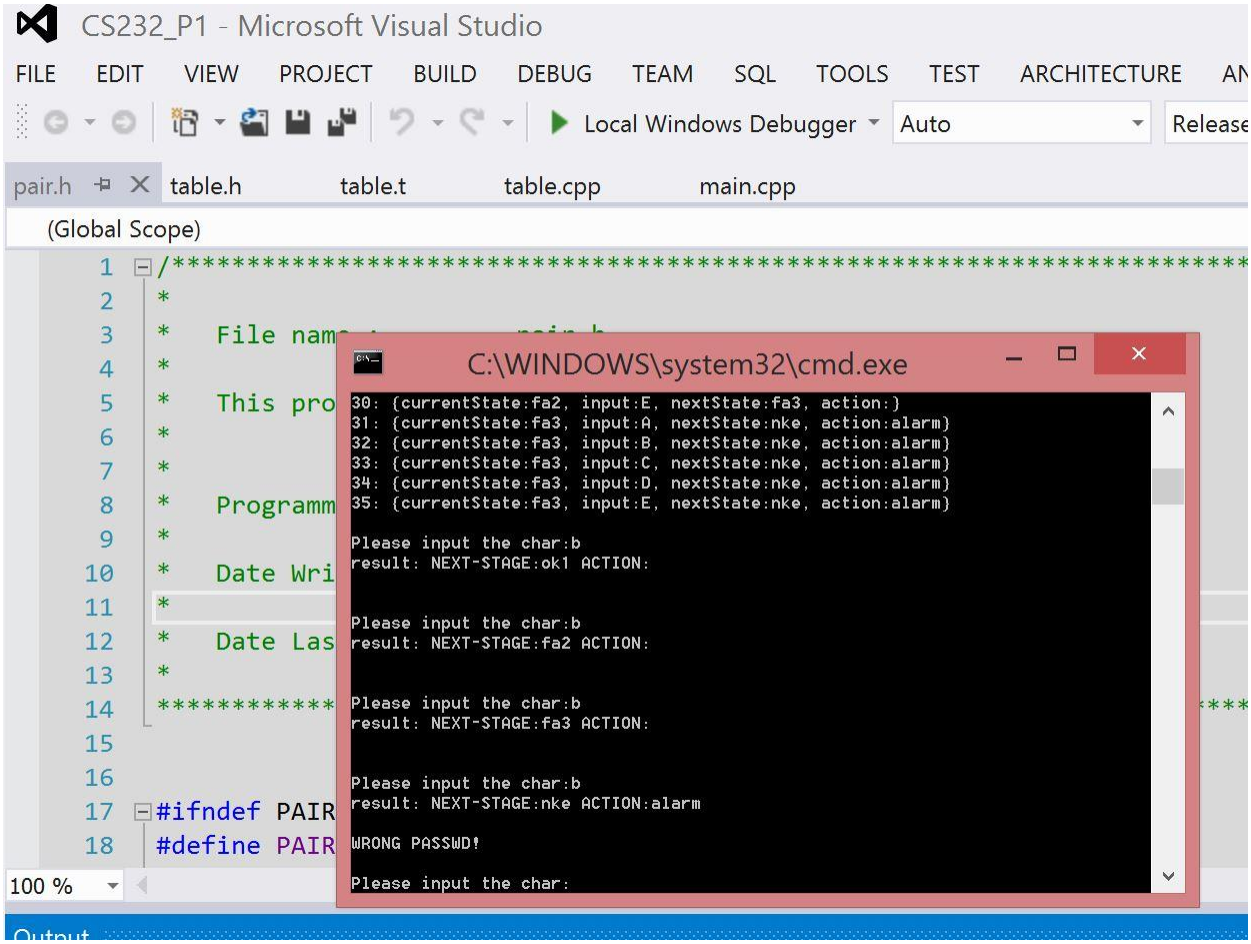
Limitations and Suggestions

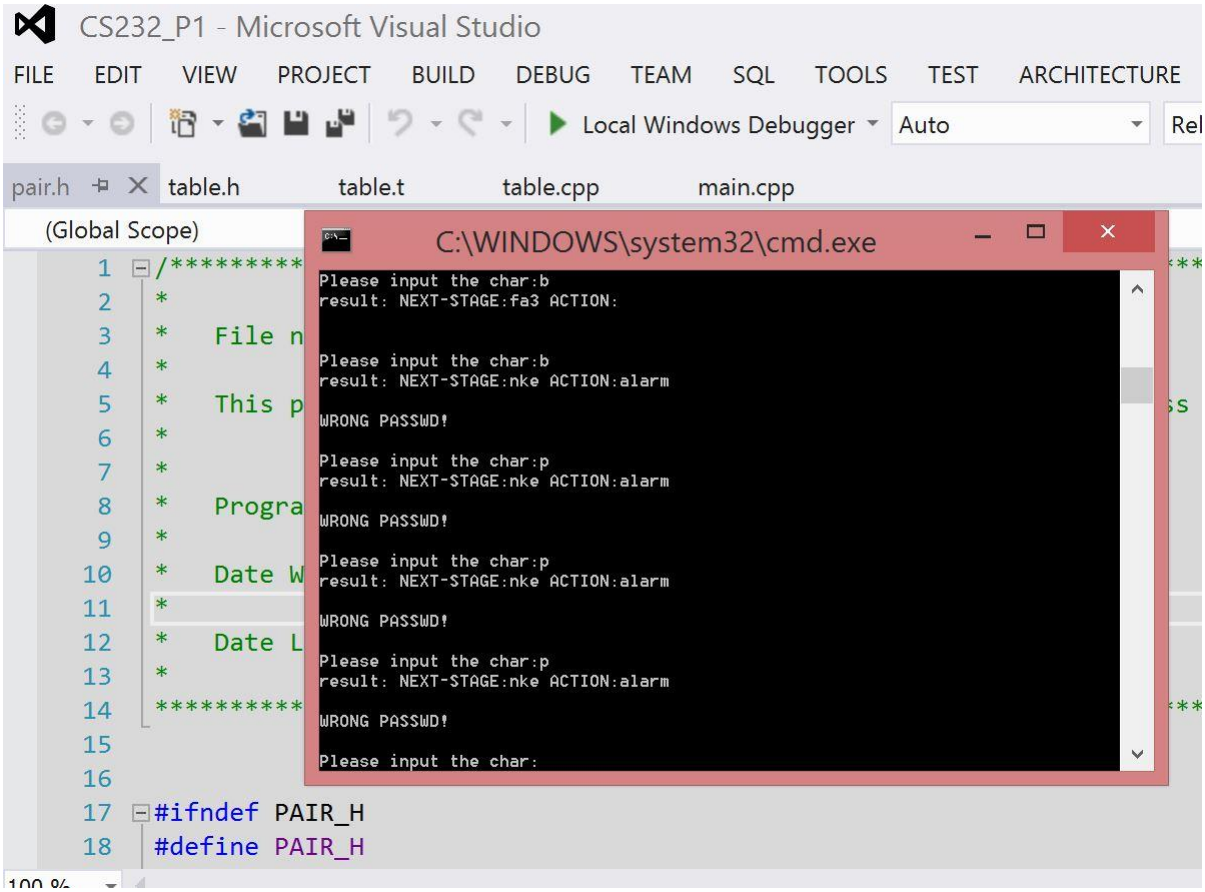
All of the classes of this program could be rewritten to be more robust and dynamic. For instance, the reading of txt class could be modified to include a coordinate for two-dimensional positions as well as other types of Positions, but the classes are sufficient for the purposes of this

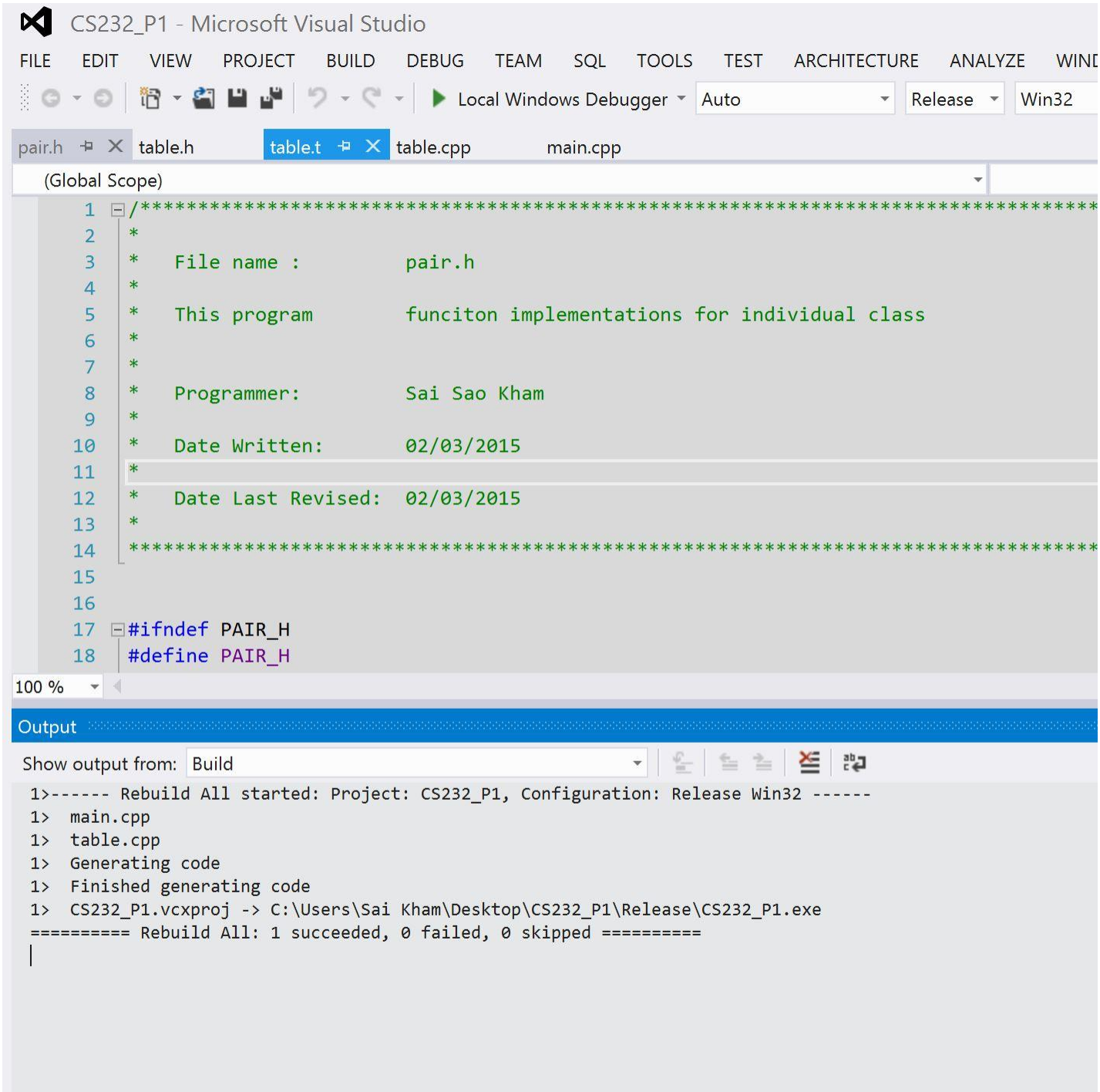
program. If someone or I were to rewrite this program I would suggest that you use an integer array rather than a character array so that the program could be more easily adapted to GUI graphics rather than ASCII art.




CS232_P1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE
Local Windows Debugger
Auto
pair.h
table.h
table.t
table.cpp
main.cpp
(Global Scope)
C:\WINDOWS\system32\cmd.exe
28: {currentState:fa2, input:C, nextState:fa3, action:}
29: {currentState:fa2, input:D, nextState:fa3, action:}
30: {currentState:fa2, input:E, nextState:fa3, action:}
31: {currentState:fa3, input:A, nextState:nke, action:alarm}
32: {currentState:fa3, input:B, nextState:nke, action:alarm}
33: {currentState:fa3, input:C, nextState:nke, action:alarm}
34: {currentState:fa3, input:D, nextState:nke, action:alarm}
35: {currentState:fa3, input:E, nextState:nke, action:alarm}
Please input the char:b
result: NEXT-STAGE:ok1 ACTION:
Please input the char:a
result: NEXT-STAGE:ok2 ACTION:
Please input the char:a
result: NEXT-STAGE:ok3 ACTION:
Please input the char:d
result: NEXT-STAGE:nke ACTION:unlock
UNLOCK!
#define PAIR_H


CS232_P1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE
Local Windows Debugger
Auto
pair.h
table.h
table.t
table.cpp
main.cpp
(Global Scope)
C:\WINDOWS\system32\cmd.exe
30: {currentState:fa2, input:E, nextState:fa3, action:}
31: {currentState:fa3, input:A, nextState:nke, action:alarm}
32: {currentState:fa3, input:B, nextState:nke, action:alarm}
33: {currentState:fa3, input:C, nextState:nke, action:alarm}
34: {currentState:fa3, input:D, nextState:nke, action:alarm}
35: {currentState:fa3, input:E, nextState:nke, action:alarm}
Please input the char:b
result: NEXT-STAGE:ok1 ACTION:
Please input the char:b
result: NEXT-STAGE:fa2 ACTION:
Please input the char:b
result: NEXT-STAGE:fa3 ACTION:
Please input the char:b
result: NEXT-STAGE:nke ACTION:alarm
WRONG PASSWD!
Please input the char:
100 %


CS232_P1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE
Local Windows Debugger
Auto
pair.h
table.h
table.t
table.cpp
main.cpp
(Global Scope)
C:\WINDOWS\system32\cmd.exe
Please input the char:b
result: NEXT-STAGE:fa3 ACTION:
Please input the char:b
result: NEXT-STAGE:nke ACTION:alarm
WRONG PASSWD!
Please input the char:p
result: NEXT-STAGE:nke ACTION:alarm
WRONG PASSWD!
Please input the char:p
result: NEXT-STAGE:nke ACTION:alarm
WRONG PASSWD!
Please input the char:p
result: NEXT-STAGE:nke ACTION:alarm
WRONG PASSWD!
Please input the char:
#ifndef PAIR_H
#define PAIR_H


CS232_P1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WIND
Local Windows Debugger
Auto
Release
Win32
pair.h
table.h
table.t
table.cpp
main.cpp
(Global Scope)
1 /***
2 *
3 * File name : pair.h
4 *
5 * This program funciton implementations for individual class
6 *
7 *
8 * Programmer: Sai Sao Kham
9 *
10 * Date Written: 02/03/2015
11 *
12 * Date Last Revised: 02/03/2015
13 *
14 ***
15
16
17 #ifndef PAIR_H
18 #define PAIR_H
100 %
Output
Show output from: Build
1>------ Rebuild All started: Project: CS232_P1, Configuration: Release Win32 ------
1> main.cpp
1> table.cpp
1> Generating code
1> Finished generating code
1> CS232_P1.vcxproj -> C:\Users\Sai Kham\Desktop\CS232_P1\Release\CS232_P1.exe
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========