# Summer 2015

# **Wedding Planner Application**

OOP244 Assignment

**Milestone 5: the Wplanner class V1.0**

Your final milestone for this project is to create an application class called "WPlanner".

Please download the source files provided for this milestone. Overwrite the classes with the ones you have developed in the four previous milestones and then start working on the WPlanner class.

WPlanner uses your previously created classes to store and retrieve OnShelf and CustomMade Good information within a file. 

## **WPlanner Class**

### WPlanner has several private member functions and only two public ones. The descriptions below specify what these functions do. For the more complicated function the description includes suggested pseudo code, which you may implement directly. Feel free to develop your own logic or to follow this pseudo code, OR to follow this pseudo code and then to modify your own logic. Save your class definition in a file named WPlanner.h and your implementation in a file named WPlanner.cpp.

### Private member variables:

#### char \_filename[256];

This variable holds the name of the text file that stores the Goods' information.

#### Good\* \_good [MAX\_NO\_RECS];

An array of pointers to **Good** objects. This array holds MAX\_NO\_RECS addresses. (i.e., each element is a pointer to one **Good** object)

#### std::fstream datafile;

An instance of an **fstream** class that refers to a file.

#### int \_noOfGoods;

The variable holds the number of **Good**s (**OnShelf** or **CustomMade**) pointed to by the **\_good** array.

### Constructor:

The WPlanner constructor receives the address of an unmodifiable char string called **filename** and then:   
1 - copies **filename** to the **\_filename** member variable  
2 - sets all the **\_good** to **nullptr**  
3 - sets **\_noOfGoods** to 0  
4 - loads the records, if any, from the file

### Private member functions:

#### Copy and assignment prevention

Make sure that a **WPlanner** object cannot be copied or assigned to another **WPlanner** object.

#### int menu()

This function displays the menu as follows and waits for the user to select an option.

Wedding Planner Management Program

1 - List goods

2 - Add On Shelf Good

3 - Add Custom-Made Good

4 - Update Good quantity

0 - Exit program

>

If the user’s selection is valid, this function returns the selection  
If the user’s selection is invalid, this function returns -1  
This function erases any characters in the keyboard buffer before returning control to its caller.

#### void loadRecs();

This function opens the file for reading. If the file does not exist, this function creates an empty file and exits. If the file exists, this function loads records from the file overwriting any old ones that may be pointed to by the **\_good** array.   
This function makes sure that any old records are deallocated before allocating memory for each new record and loading each records into that memory. Once this function has completed loading, it closes the file and returns control to its caller.

Pseudo code:

set readIndex to zero

open the file for reading (use ios::in)

if the file is in fail state it means there is no file on the disk, then

clear the failure

close the file

open the file for writing (ios::out) to create the file

close thefile

otherwise

until reading fails loop

if the address in the Good pointer at readindex is not nullptr, delete the memory at that address

read one into Id character

if Id character is C

Dynamically create a CustomMade object and store its address in the Good pointer at readIndex

if Id character is S

Dynamically create an OnShelf object and store its address in the Good pointer at readIndex

if either C or S is read

skip the comma in the file record

load the Good from the file (using its load method)

add one to read index

continue the loop

set number of Goods to readIndex

close the datafile

#### void saveRecs()

This function opens the file for writing, loops through the **\_good** array up to **\_noOfGoods** and stores each Good in the datafile. Finally, this function closes the file.

#### void listGoods() const

First, this function prints the following title:

Row | UPC | Good Name | Cost |Tax| QTY|Need| Delivery

-----|--------|--------------------|-------|---|----|----|----------

Then, this function loops through the **\_good** array up **to \_noOfGoods** and for each Good:

1 - prints the Row number in a field of 4 right-justified  
2 - prints a Bar character (|) surrounded by two spaces

3 - prints the current Good followed by a newline

While executing the iteration, this function calculates the total cost of the Goods in a double value using the operator+= implemented for the Good class: (double operator+=(double& d, const Good& G)).

After this function has completed printing the list, it prints:

---------------------------------------------------------------------------

and then prints the total cost like this:

Total cost of the Wedding: $999.99

This function prints the total cost value with a Dollar sign at the left and two digits after the decimal point

#### int SearchGoods(const char\* upc)const

This function loops through the **\_good** array up to **\_noOfGoods** and checks each one for the same UPC as the incoming argument using the operator== implemented for the Good class.  
If this function finds a match it returns the index, otherwise it returns -1.

#### void updateQty(const char\* upc)

This function updates the quantity on hand for a particular Good. This function searches for the Good with the same upc as the incoming argument. If this function doesn’t finds a match, it displays:

"Not found!"<NEWLINE>

If this function finds a match, it displays the Good in non-linear format and asks the user for the quantity purchased in the form of an integer:

"Please enter the number of purchased goods: "

If this function cannot read the integer it prints:

"Invalid Quantity value!"<NEWLINE>

If this function reads the integer, it makes sure the amount specified is less than or equal to the amount required (i.e., less than qtyNeeded() - quantity()). If the amount specified is less than or equal to the amount required, this function adds the value to the quantity on hand using the operator+= overloaded for a Good. If the amount specified is not less than or equal the amount required, this function only accepts the amount required and prints a message to return the extra:

"Too many purchased, only 9 needed, please return the extra 7.” <NewLine>

After processing the user’s input, this function save all records back to the file and prints:

"Updated!"

Finally, this function flushes any extra characters from the keyboard buffer.

#### void addGood(bool isCustomMade)

Depending on the value **isCustomMade**, this function creates a **CustomMade** or **OnShelf** object and gets its values from the user. If an error occurs, this function displays the Good and exits the function. If no error occurs, this function opens the **datafile** for writing, stores the Good and then closes the file.

#### int run()

This function displays the menu and depending on the user’s selection, performs the requested action and redisplay the menu. This function stops displaying the menu once the user has entered zero to exit.

If the user selects 4, this function prints:

"Please enter the UPC: "

gets the UPC and updates the quantity.

If the user selects 0, this function prints:

"Goodbye!!"<NEWLINE>

and terminates the program’s execution.

If the user selects an invalid option, this function prints:

"===Invalid Selection, try again.==="

# **Submission:**

Please refer to your professor’s instructions for submission.